

# Lecture 7

# Class diagrams

11486 Systems Analysis and Modelling

6677 Systems Analysis and Modelling G

Dr Luke Nguyen-Hoan

# Contents

## Administration

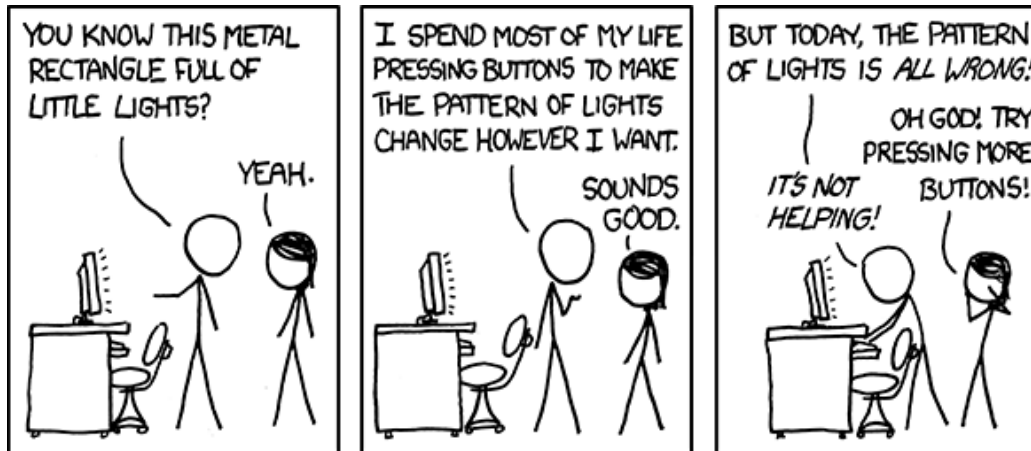
1. Classes and objects
2. Syntax and semantics
3. Advanced concepts

Textbook chapters:

Systems Analysis and Design in a Changing World  
Chapter 4 Domain Modeling  
Chapter 12 Object-Oriented Design: Fundamentals

UML@Classroom  
Chapter 4 The Class Diagram





Computer problems  
Image by Randall Munroe  
<https://xkcd.com/722/>



# Administration

# Assessment – Quiz 3 for UG students

The third quiz opens right after this lecture, and is due at 11:59pm on Sunday

No late submissions due to the nature of this assessment (as per the unit outline)

# Assessment - Group assignment

Due end of week 12

Full information is in the assignment instructions available on Canvas, including the following group formation information

# Assessment - Group assignment groups

## Choosing a group for the assignment (before Week 10 Tuesday 14/04/2020)

To sign up for a group on Canvas, follow the instructions here:

<https://community.canvaslms.com/docs/DOC-10516>

The groups for this assignment are under the “Assignment Group” category on Canvas. Each group is labelled with the case study for that group, to help you choose an appropriate group if needed or get in touch with other students to ask to work together. Make sure you choose a group that matches the case study you have chosen!

Self sign up for groups will be closed on the day listed above. If you have any issues with sign up or your group composition, please contact the unit convener as soon as possible. The longer you delay, the less that can be done to resolve any issues!

# Assessment - Group assignment groups

Groups for this assignment are limited to a maximum of 4 students per group, with no exceptions. Please note that this assignment is intended for groups of 3-4 students, so students in groups smaller than 4 students should expect that more effort and time is required to accomplish equivalent grades. No consideration will be given for groups comprising less than 3 students.

Undergraduate students (11486) should form a group with other undergraduate students only.

Graduate students (6677) should form a group with other graduate students only.

Note that there is no requirement for all group members to be in the same tutorial – you may form a group with friends from other tutorials if you wish. However, it may be easier for you to coordinate and work together if you form a group with other people in your tutorial.

# Assessment - Group assignment groups

Note that there is no requirement for all group members to be in the same tutorial – you may form a group with friends from other tutorials if you wish. However, it may be easier for you to coordinate and work together if you form a group with other people in your tutorial.

It is each student's responsibility to organise their group – the tutors and unit conveners will not assist you in finding or joining a group. You may wish to use the Canvas Discussion forums to help find a group.

If there are no empty groups remaining for the case study that your group has chosen, email the unit convener and additional groups will be added.

# Assessment - Group assignment groups

## Groups finalised for the assignment – Week 11 Monday 20/04/2020

This is just under two weeks prior to the assignment due date, and thus by this point everyone should now be in groups and have organised to complete the assignment with their peers.

Any students not in a group by this date will be assumed to be completing the assignment individually, and will thus be placed into a group of 1 student only.

## Any changes requested after this date will result in a mark penalty of 10% of the available marks for the students involved

This deadline is imposed as changes to group allocations after assignments have started to be submitted can cause issues with Canvas submissions and grading, so groups need to be finalised in advance of any assignments being submitted.

# Interface student experience questionnaire (ISEQ)

Confidential – I never see who made a comment or response

Please answer! I really do read all of your comments.

If you think something's good or you like something – say it, so I know to keep doing it and I won't change it!

If you think something's bad or you don't like something – say it, so I can take your feedback on board!

UNIVERSITY OF  
CANBERRA

## YOUR VOICE IS IMPORTANT

When teachers are given feedback, we can identify what is working well, and what needs to be improved. To help us do this, please keep your feedback:

- Specific and respectful
- Constructive and actionable



# Questions?

Any questions about the unit before we get into this week's topic?

- Assessment
- Tutorials
- Anything else?



# 1. Classes and objects

- a) Big picture
- b) Objects
- c) Classes
- d) Classes and objects
- e) Example

# Big Picture

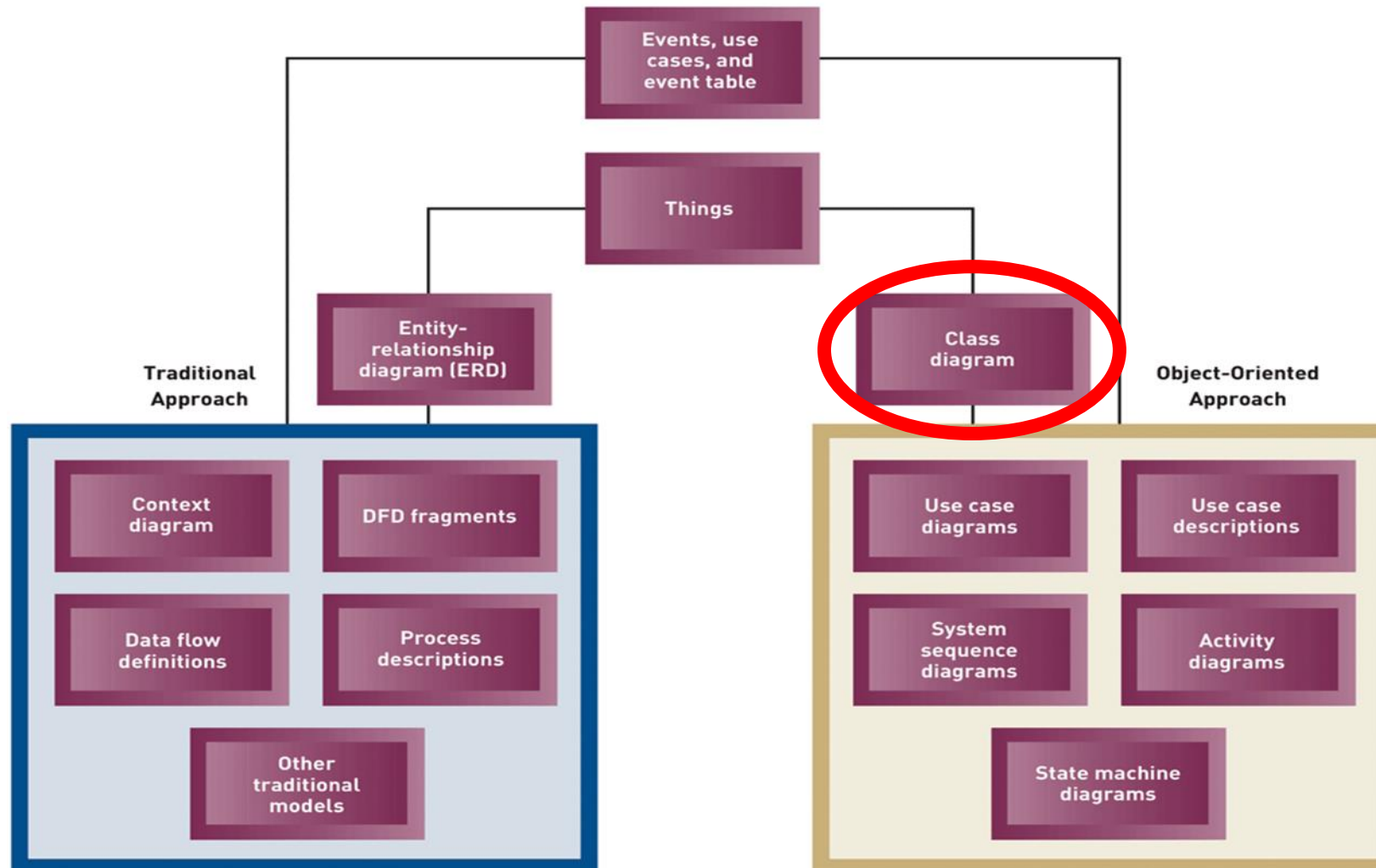


Diagram from Systems Analysis and Design in a Changing World, J. Satzinger, R. Jackson, S. Burd. Used with permission.

# Objects

An object is something that has at least one of:

- Identity

- State

- Behaviour

*(In many cases, has all three)*

Objects can be physical or conceptual



# Classes



A class is a set of objects that have the same type of properties, state, and/or behaviour

# From programming

If you've learnt about classes and objects from programming, in analysis we mostly consider the public interfaces of classes and objects, as these are what drive interaction and behaviour

Some elements that are commonly programmed as private are still included in object-oriented analysis (mostly variables that when implemented may not be directly accessed)



# Classes and objects



A class is a set of objects that have the same type of properties, state, and/or behaviour

Can be thought of as a template for objects

“All objects of this class have these properties, is in one of these potential states, can perform potential behaviours...”

# Classes and objects



An object is an *instance* of a class

An object will have actual values attached to it, be in one particular state, etc

For example: “Alice’s bank account” is an object, and is an instance of the “bank account” class

The *type* of “Alice’s bank account” is “bank account”

# Classes and objects

Class: Bank account

Object: Alice's bank account

# Classes and objects

Class: Bank account

Object: Alice's bank account

Linked to a particular customer

Has a balance

Has a type of savings or credit account

The following actions can be performed on an account:

- Withdrawal

- Deposit

- Close account

- Generate statement

# Classes and objects

Class: Bank account

Linked to a particular customer

Has a balance

Has a type of savings or credit account

The following actions can be performed on an account:

- Withdrawal

- Deposit

- Close account

- Generate statement

Object: Alice's bank account

Linked to Alice

Has a balance of \$1640.56

Type is a savings account

The following actions can be performed on Alice's account:

- Withdrawal

- Deposit

- Close account

- Generate statement

# Classes and objects

Class: Bank customer

Object: Alice

# Classes and objects

Class: Bank customer

Object: Alice

Has a first name

Has a last name

Has a date of birth

Has an address

Has a contact email address or  
phone number

Linked to some accounts

Details can be updated

Can open a new account

# Classes and objects

Class: Bank customer

Has a first name

Has a last name

Has a date of birth

Has an address

Has a contact email address or  
phone number

Linked to some accounts

Details can be updated

Can open a new account

Object: Alice

First name: Alice

Last name: Smith

Date of Birth: 01/01/1990

Address: 123 Somestreet St,  
Somesuburb, ACT 1234, Australia

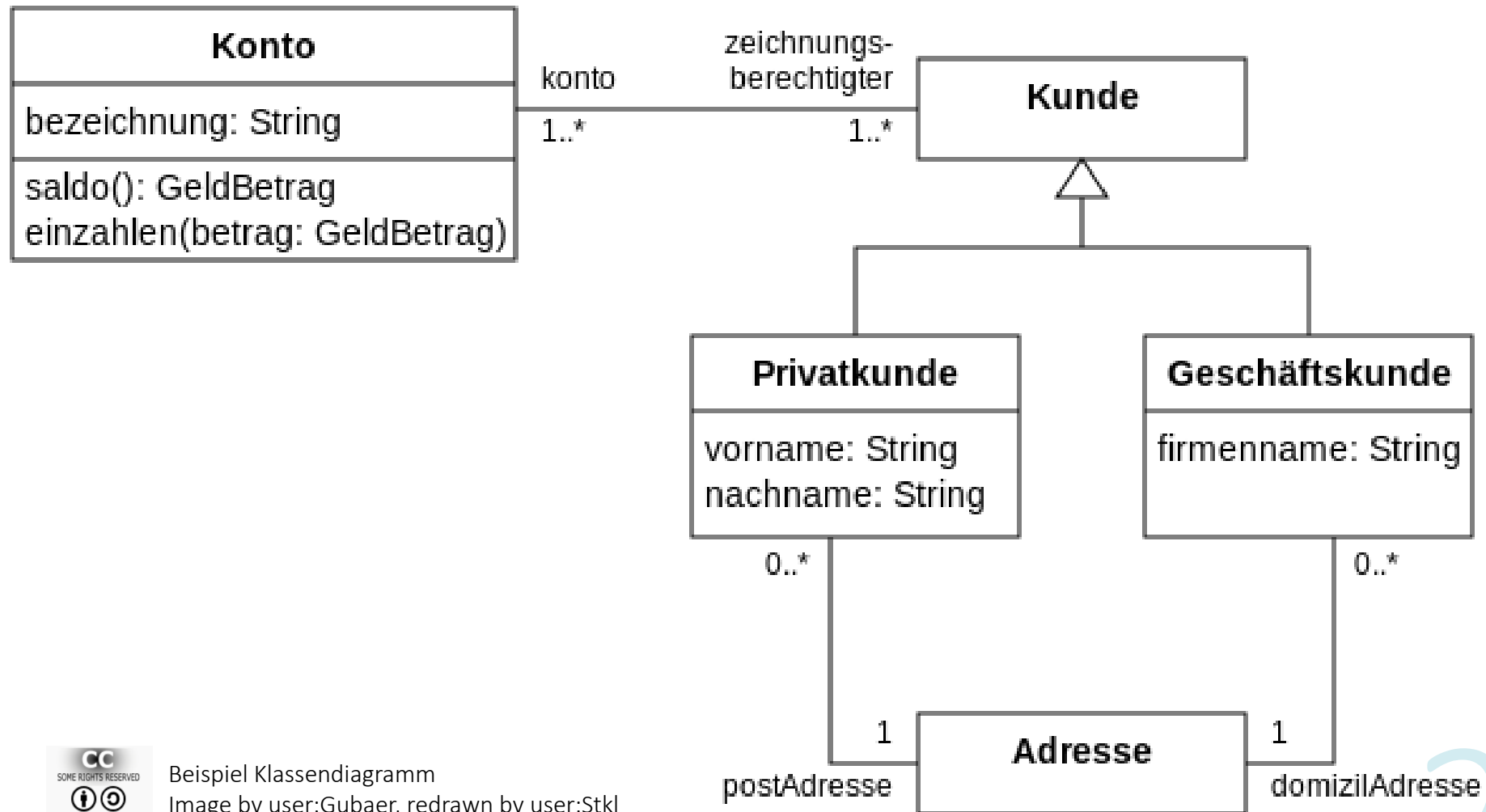
Contact: alice.smith@gmail.com

Linked to Alice's bank account

Details can be updated

Can open a new account

# Example

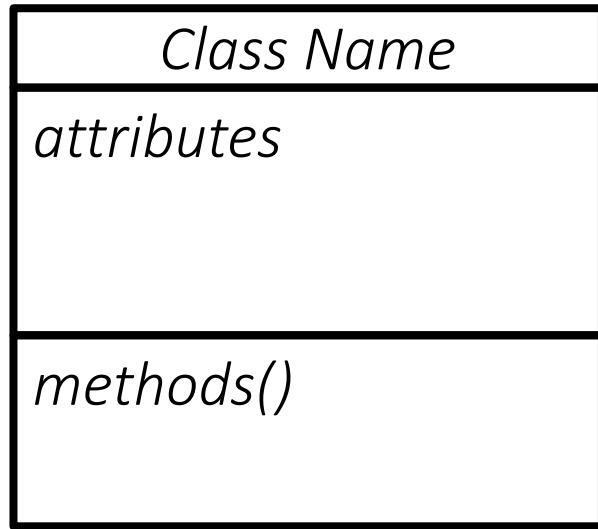




## 2. Syntax and semantics

- a) Syntax (notation)
- b) Classes
- c) Associations
- d) Attributes and methods

# Syntax (notation)



Class with attributes and methods



Association with verb phrases and multiplicities



## 2b. Classes

# Classes

| <i>Class Name</i> |
|-------------------|
| <i>attributes</i> |
| <i>methods()</i>  |

Classes are named with a noun or noun phrase

Singular (“Customer”, not “Customers”)

A class that represents a collection of things should still be singular (“Customer group”, not “Customers”)

Always capitalised

Do not use “A Customer”, “The Customer”, etc.

# Classes

| <i>Class Name</i> |
|-------------------|
| <i>attributes</i> |
| <i>methods()</i>  |

We'll get to attributes and methods later in this lecture

# Classes



*Class Name*

Sometimes, you may see a class diagram (or classes) that do not include the Attribute and Method boxes – this is still correct

This means that either there are no attributes or methods, or that the attributes and methods have not yet been considered (may not be important at the current level of abstraction)

# Classes

*Class Name*

In this unit's assessment, do not use this - use the full three compartment notation

However, it's fine to use this single box notation in the tutorials and in your work-in-progress

I will be using this single box notation in this lecture where attributes and methods aren't important, because it uses less space on the slide

Sometimes, you may see a class diagram (or classes) that do not include the Attribute and Method boxes – this is still correct

This means that either there are no attributes or methods, or that the attributes and methods have not yet been considered (may not be important at the current level of abstraction)

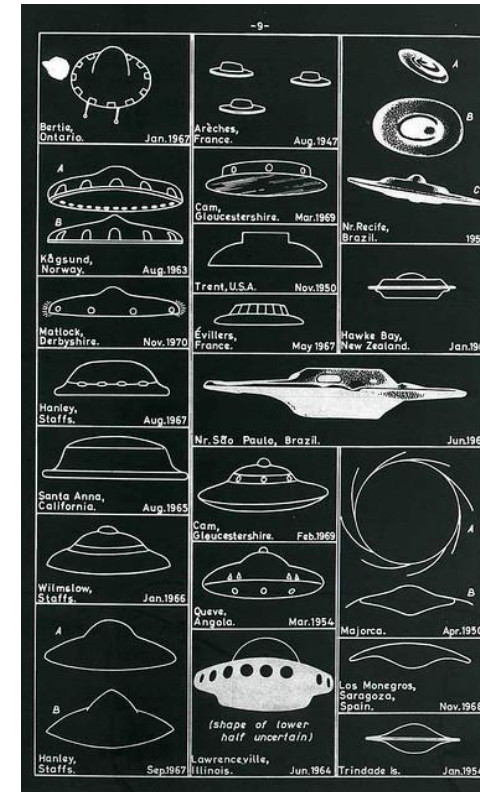
# Identifying classes

List nouns (things) from the system

Remember, objects can be physical or conceptual

So classes can represent physical things (actors, etc) or conceptual things (data records, organisational units, etc)

Also remember that classes are *types* of things, whereas objects are particular *instances* or examples of things



# Identifying classes

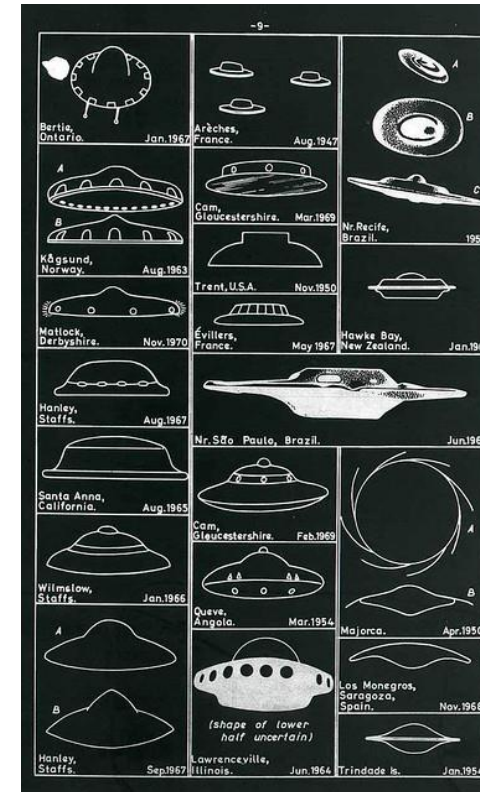
Beware that some nouns might not be classes, but instead might be objects (instances of classes) or attributes instead

For example:

“Customer” is probably a class

“John Smith” is probably an object

“Date of Birth” is probably an attribute



# Classes

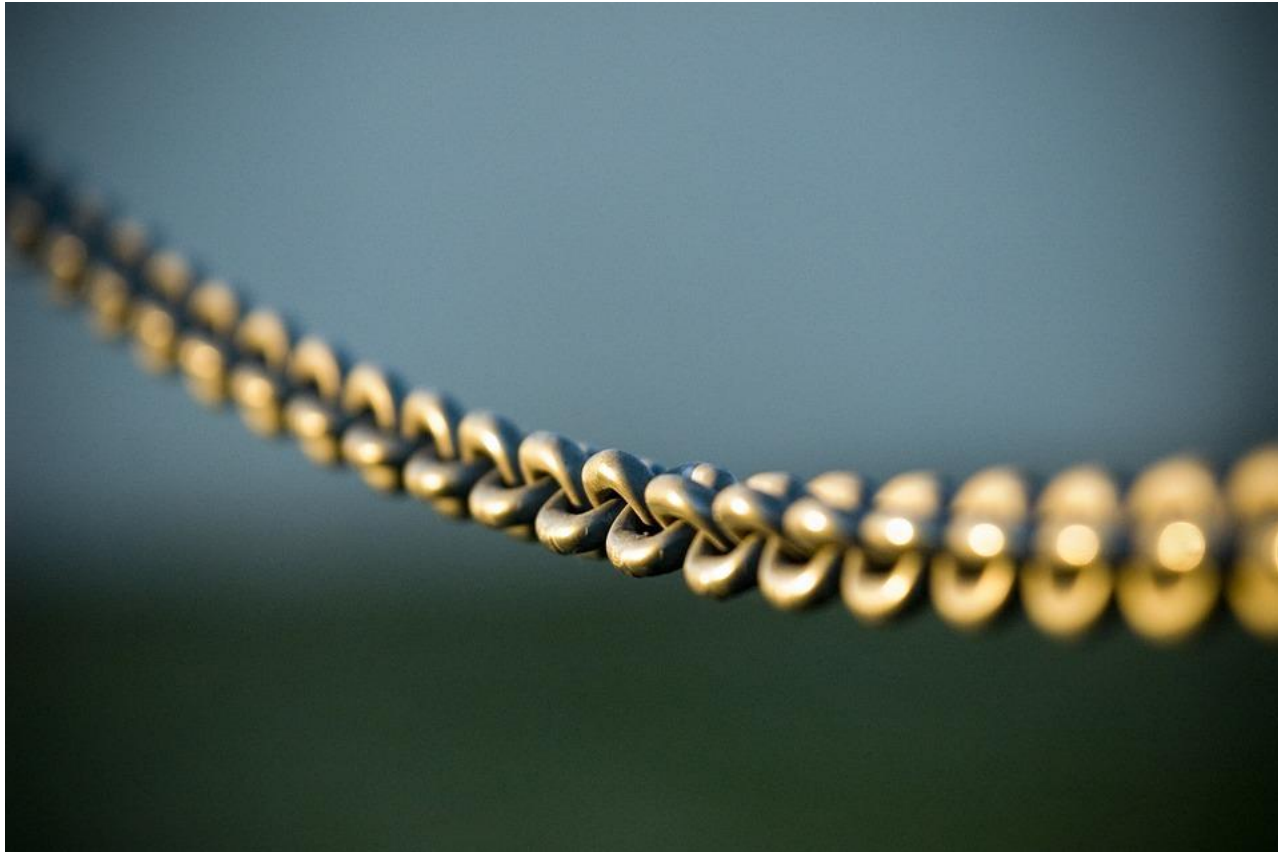
Here are some potential classes in the Bank Account Management System

Bank Account

Customer

Transaction

Statement



## 2c. Associations

- a) Associations
- b) Roles
- c) Multiplicities
- d) Reading associations
- e) Reflexive associations
- f) Multiple associations

# Associations

Associations are relationships between instances of a class (these are similar to relationships in Entity-Relationship Diagrams)

Bank Account

Customer

Transaction

Statement

# Associations

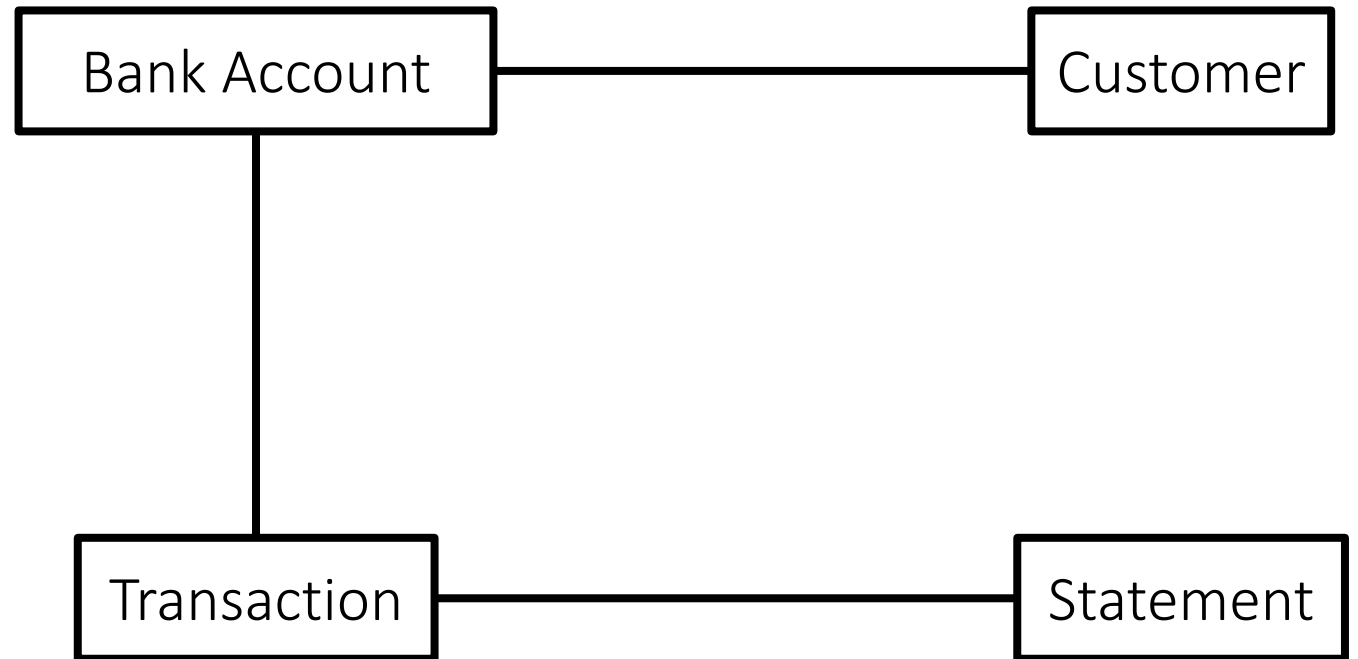
Associations are relationships between instances of a class

*A Bank Account is owned by an Existing Customer*

*A Transaction is for a Bank Account*

*A Statement is a list of Transaction(s)*

These are represented by lines joining the classes



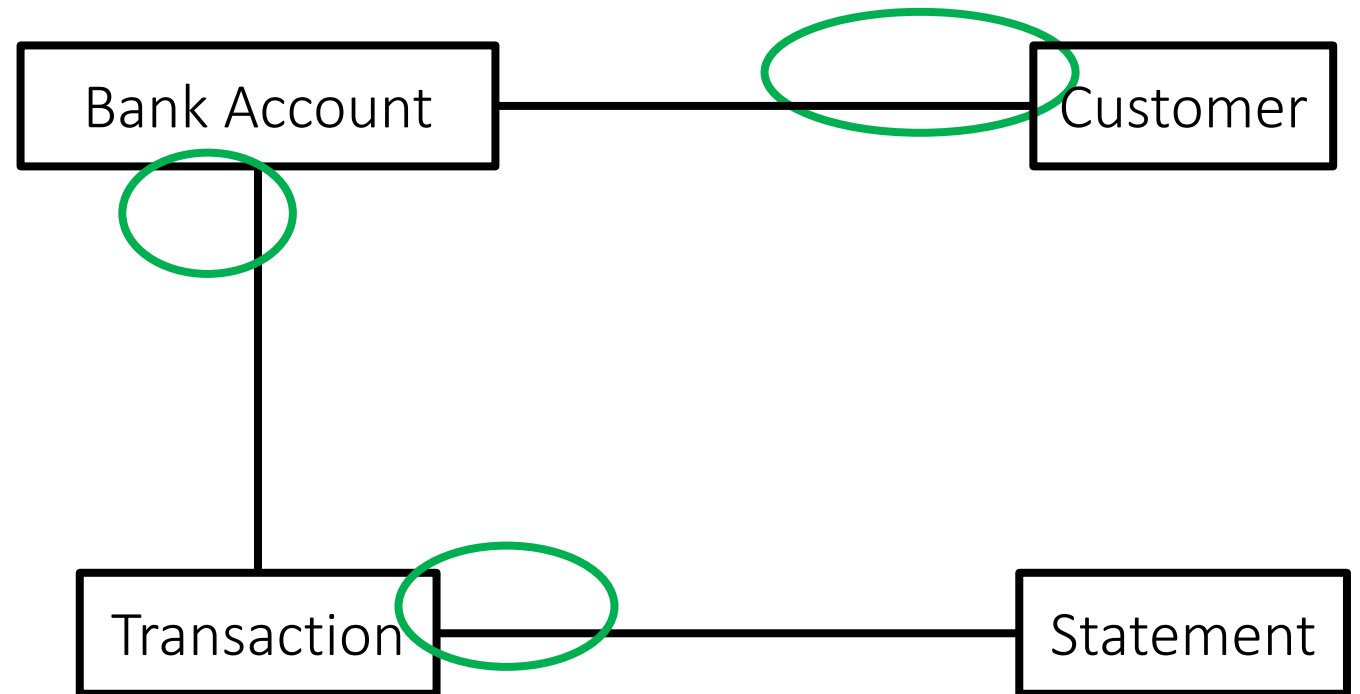
# Roles

Roles describe the meaning of an association between classes

*A Bank Account is owned by an Existing Customer*

*A Transaction is for a Bank Account*

*A Statement is a list of Transaction(s)*



# Roles

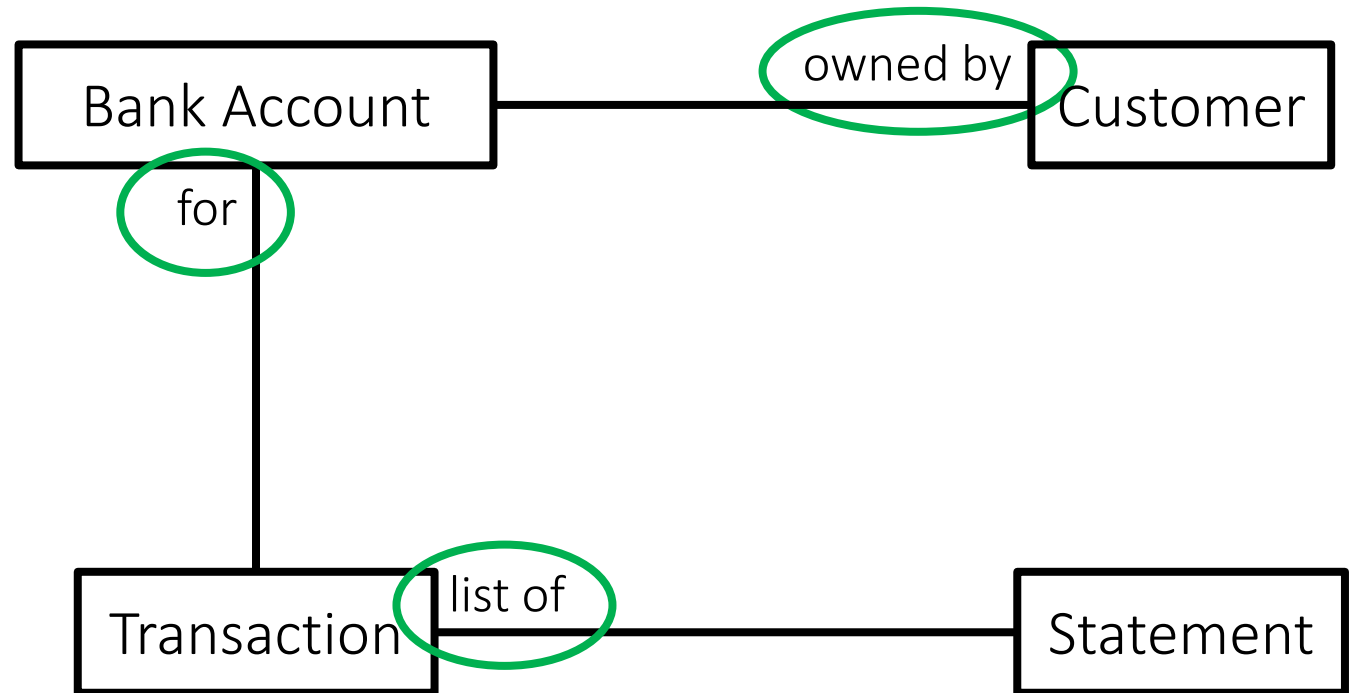
Roles describe the meaning of an association between classes

*A Bank Account is owned by an Existing Customer*

*A Transaction is for a Bank Account*

*A Statement is a list of Transaction(s)*

These are represented by *verb phrases* on both ends of the association line



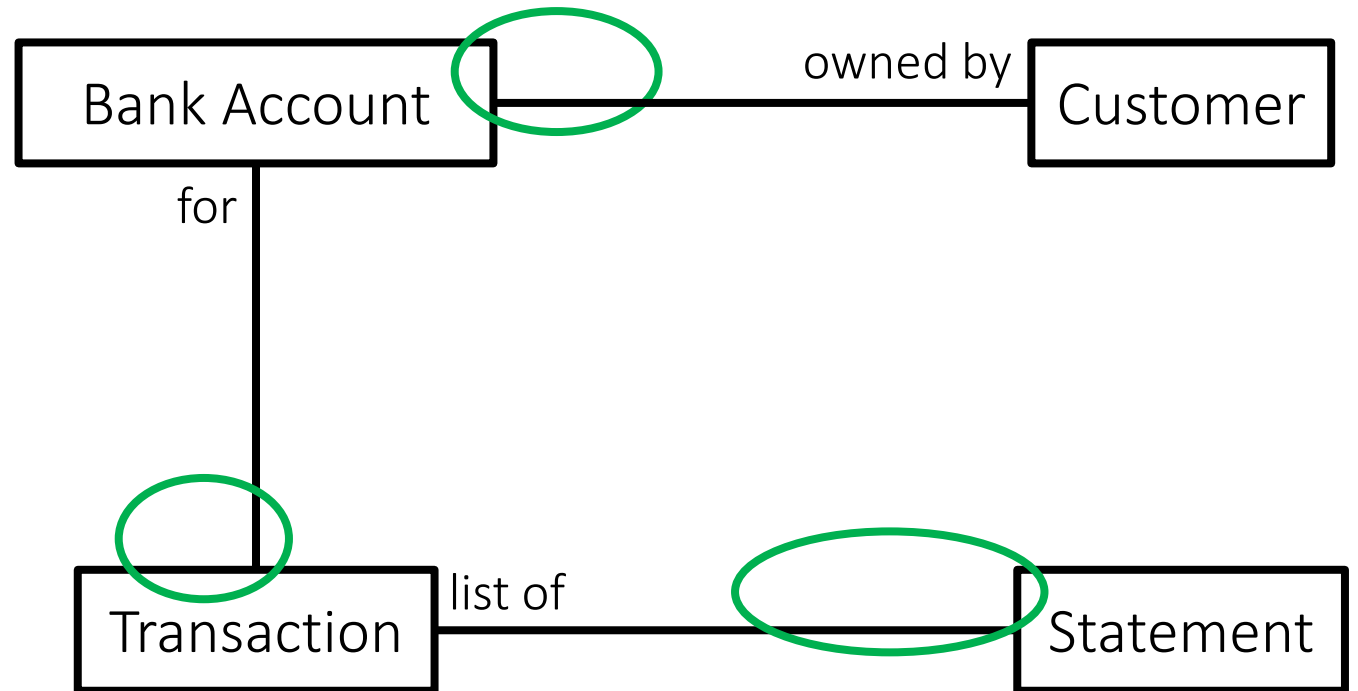
# Roles

Roles go both ways:

*An Existing Customer ??? a Bank Account*

*A Bank Account ??? a Transaction Record*

*A Transaction ??? a Statement*



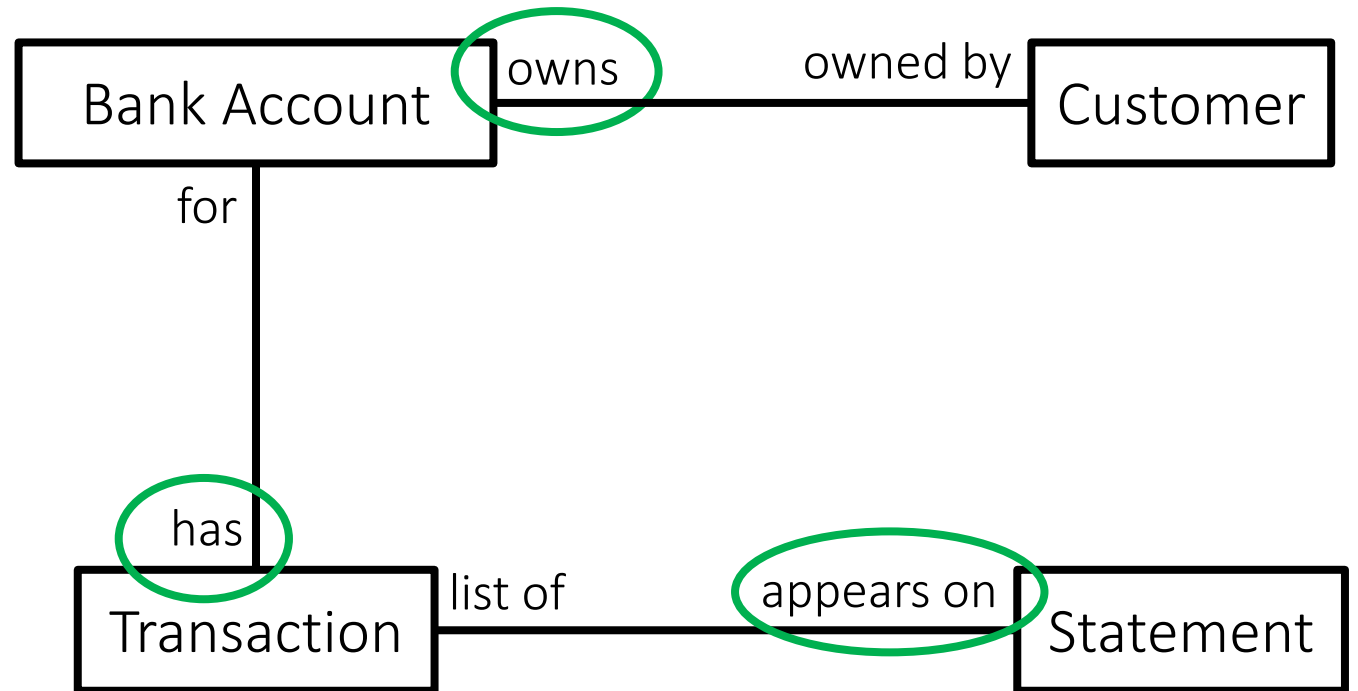
# Roles

Roles go both ways:

An *Existing Customer* owns a Bank Account

A *Bank Account* has a *Transaction Record*

A *Transaction* appears on a *Statement*

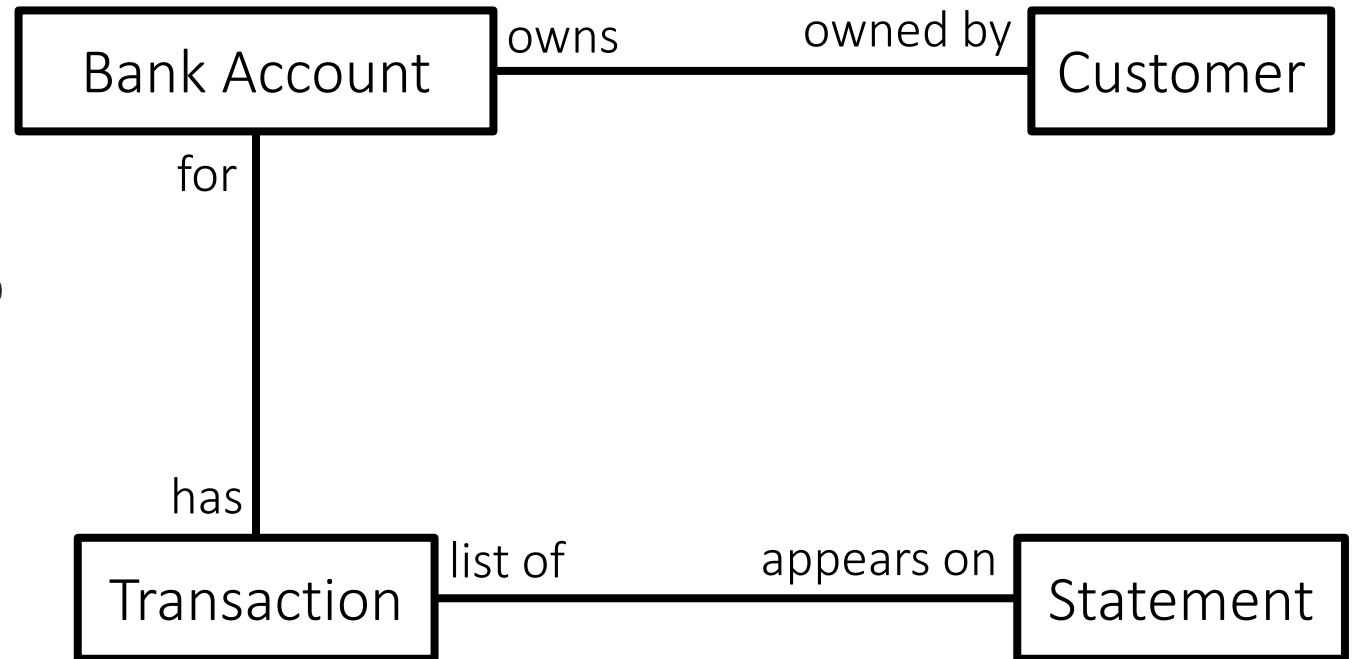


# Roles

Try to make roles as descriptive as possible

In this example, “for” and “has” are not very descriptive

Can you think of any better verbs to describe the association between *Bank Account* and *Transaction*?

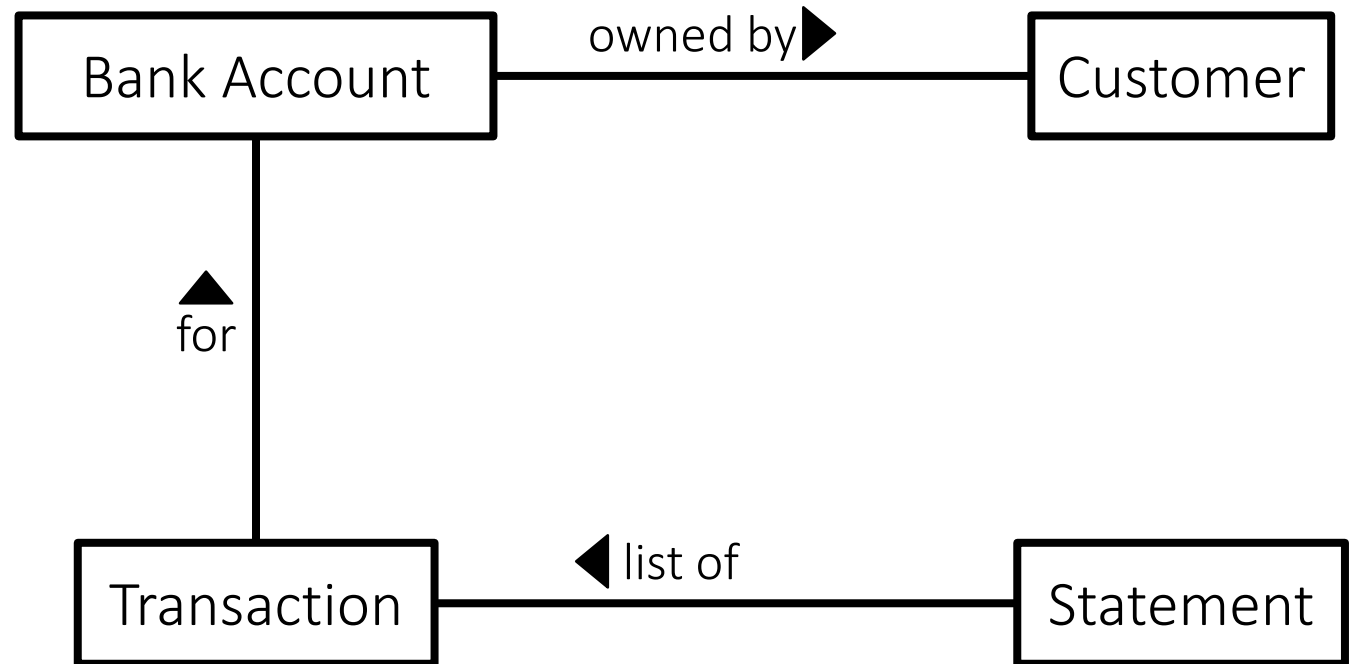


# Roles – alternative notation

You can use a single role verb instead of the double-ended notation

The arrows are important, so that readers know which way to read the association role

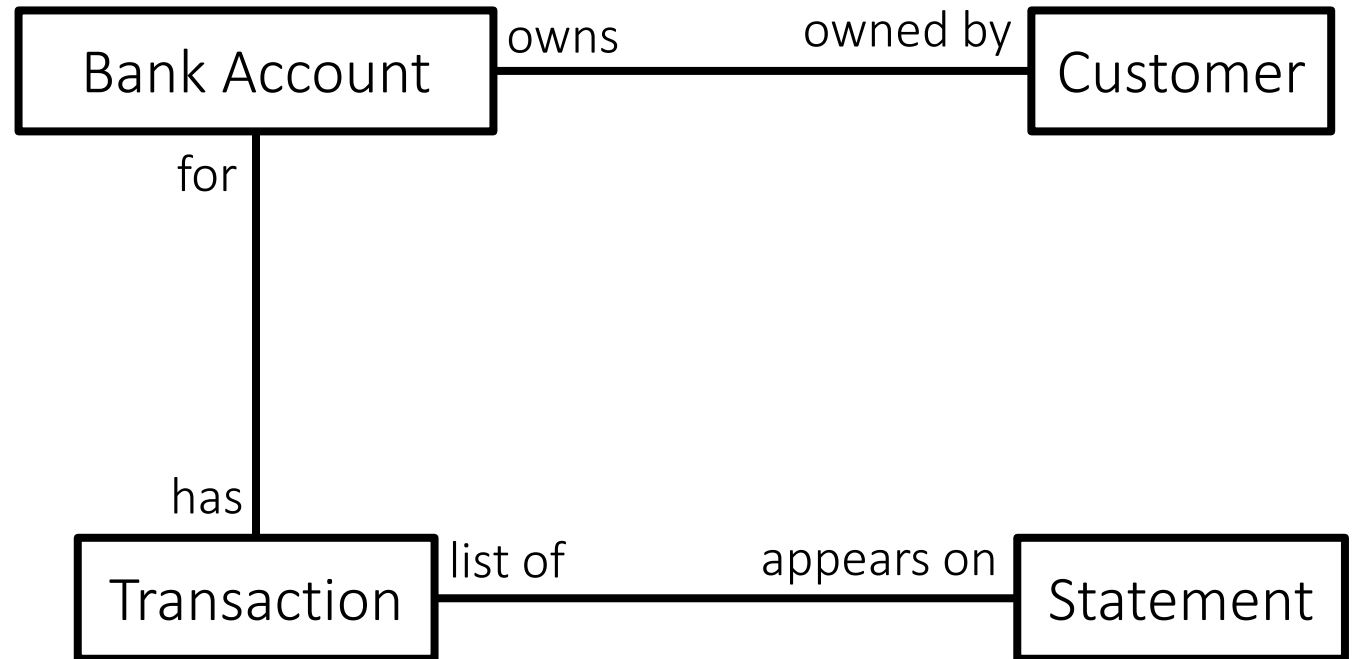
I'll keep using the double-ended notation, but the single role verb with arrow notation is equally acceptable (make sure you include the arrow though!)



# Multiplicities

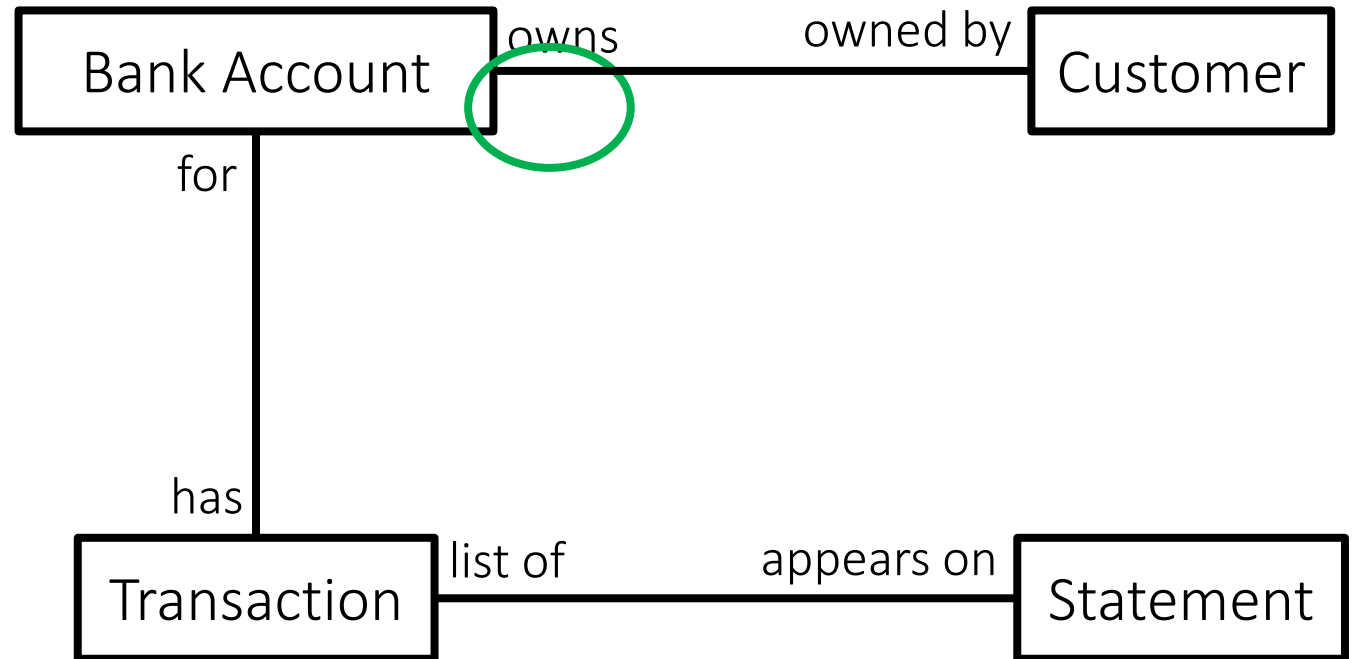
Multiplicities describe how many objects participate in an instance of a relationship

(these are similar to cardinalities in Entity-Relationship Diagrams)



# Multiplicities

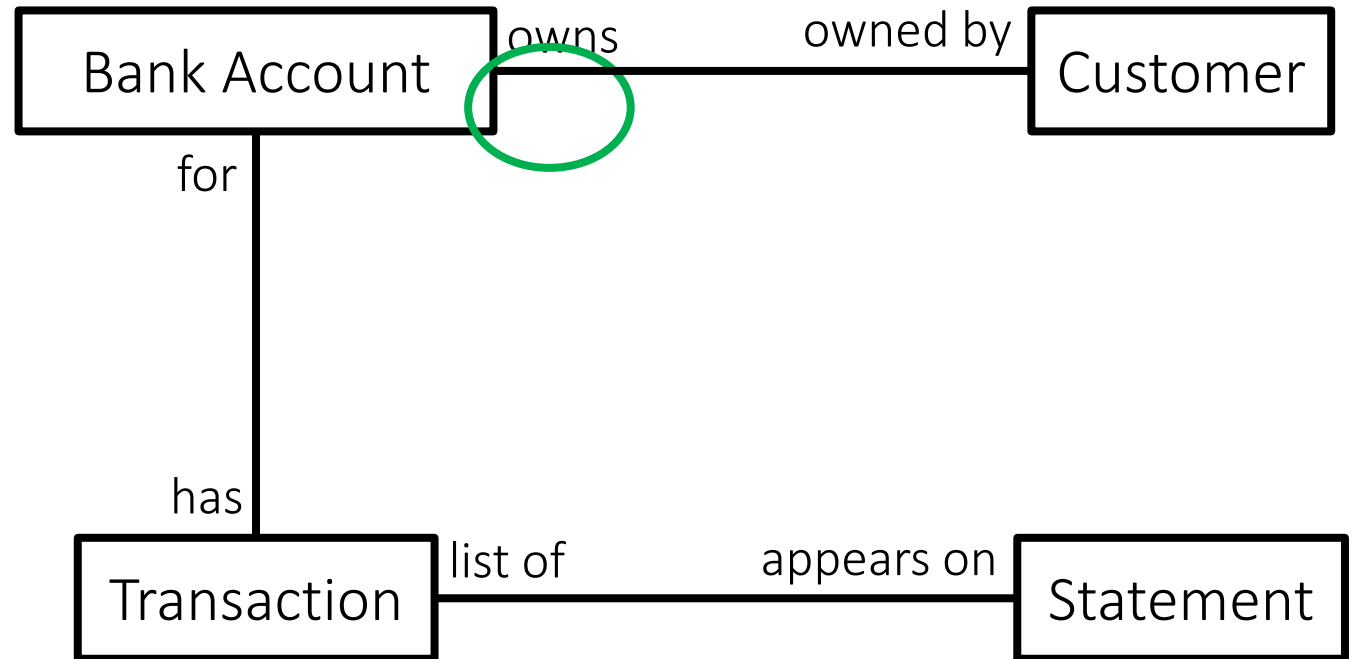
How many *Bank Account*(s) can a particular *Existing Customer* own?



# Multiplicities

A particular *Existing* Customer can own as many *Bank Account(s)* as they want

Is zero okay?



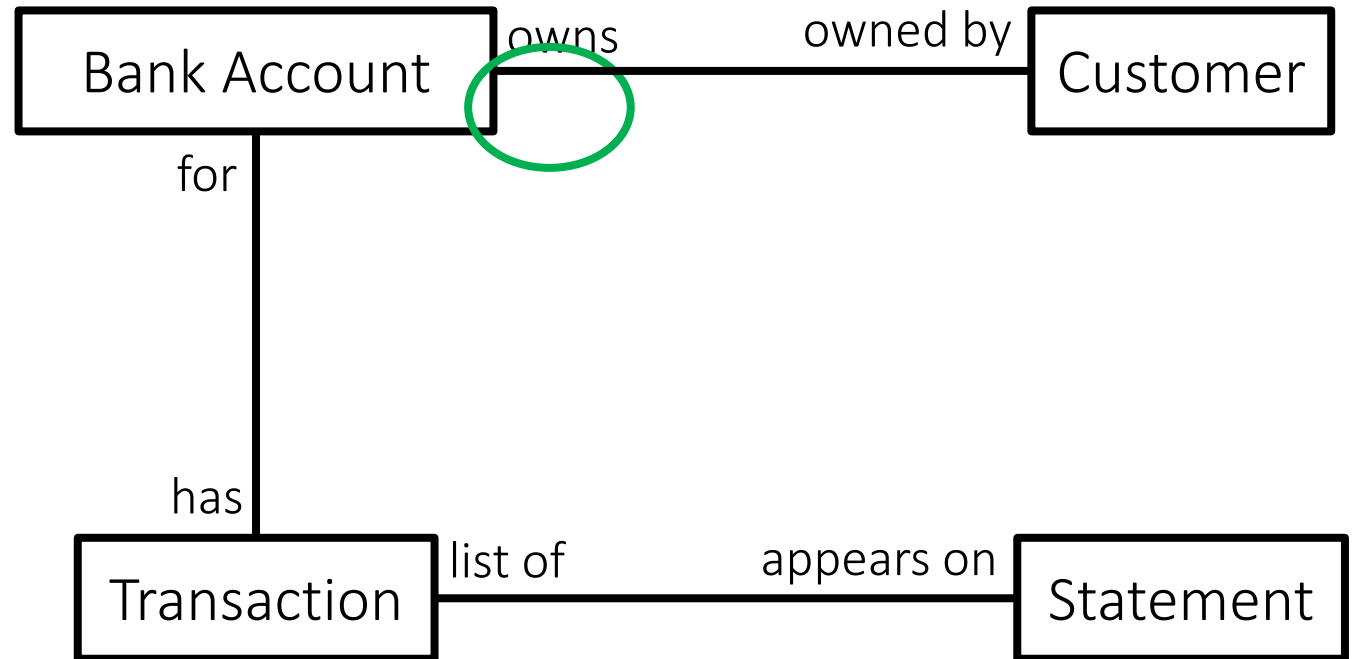
# Multiplicities

A particular *Existing* Customer can own as many *Bank Account(s)* as they want

Is zero okay?

Probably not? (if they don't have any accounts, they're not a customer?)

So a minimum of 1, maximum of as many as you like



# Multiplicities

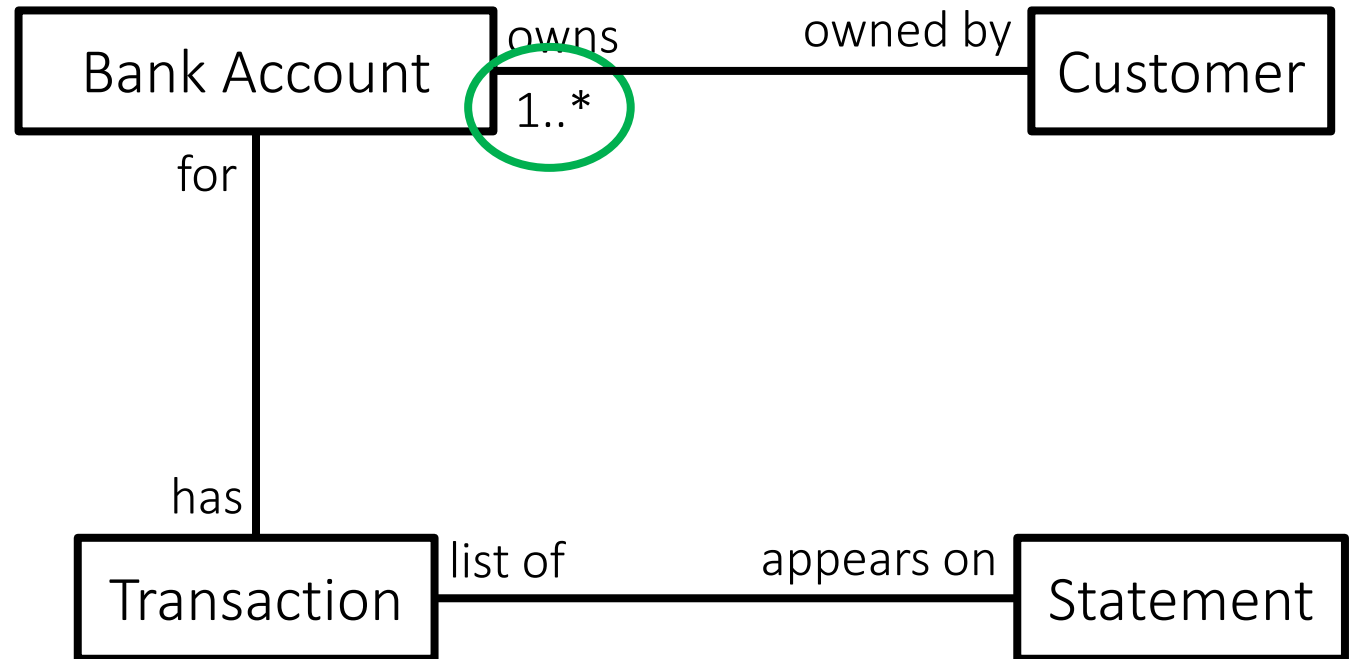
A particular *Existing* Customer can own as many *Bank Account(s)* as they want

Is zero okay?

Probably not? (if they don't have any accounts, they're not a customer?)

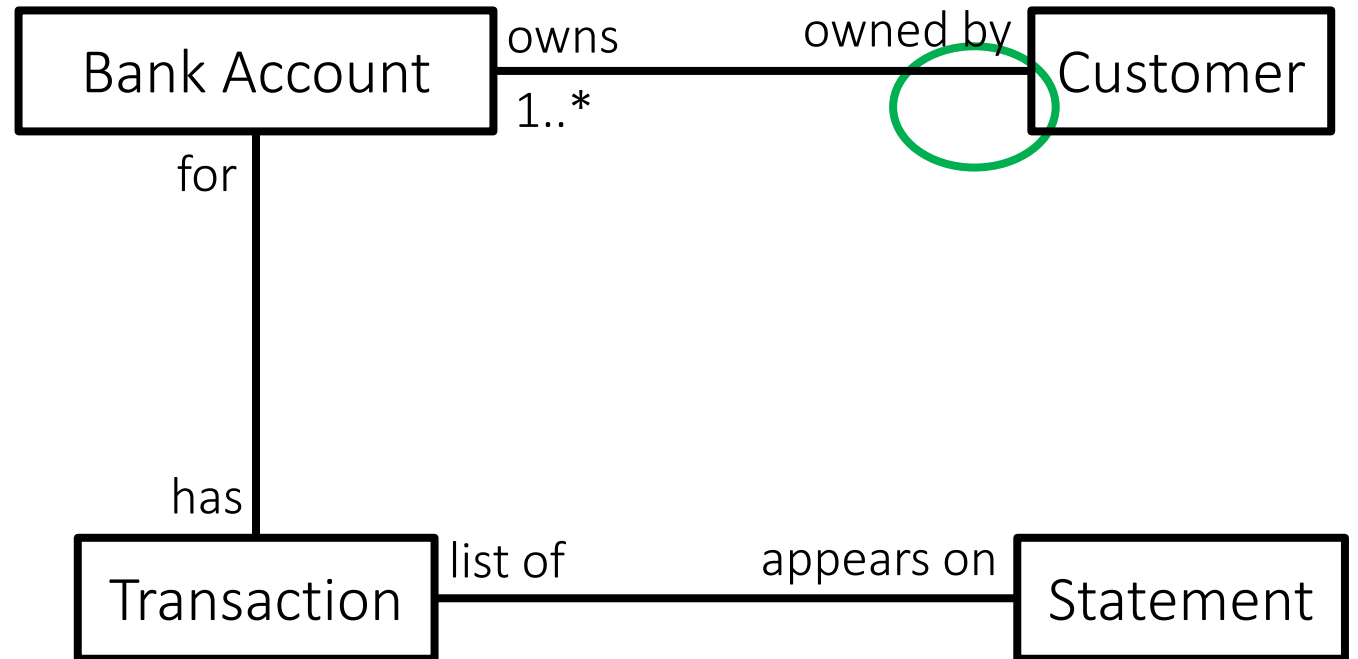
So a minimum of 1, maximum of as many as you like

Just like in data dictionaries and lists of elements, \* means as many as desired (read as "many")



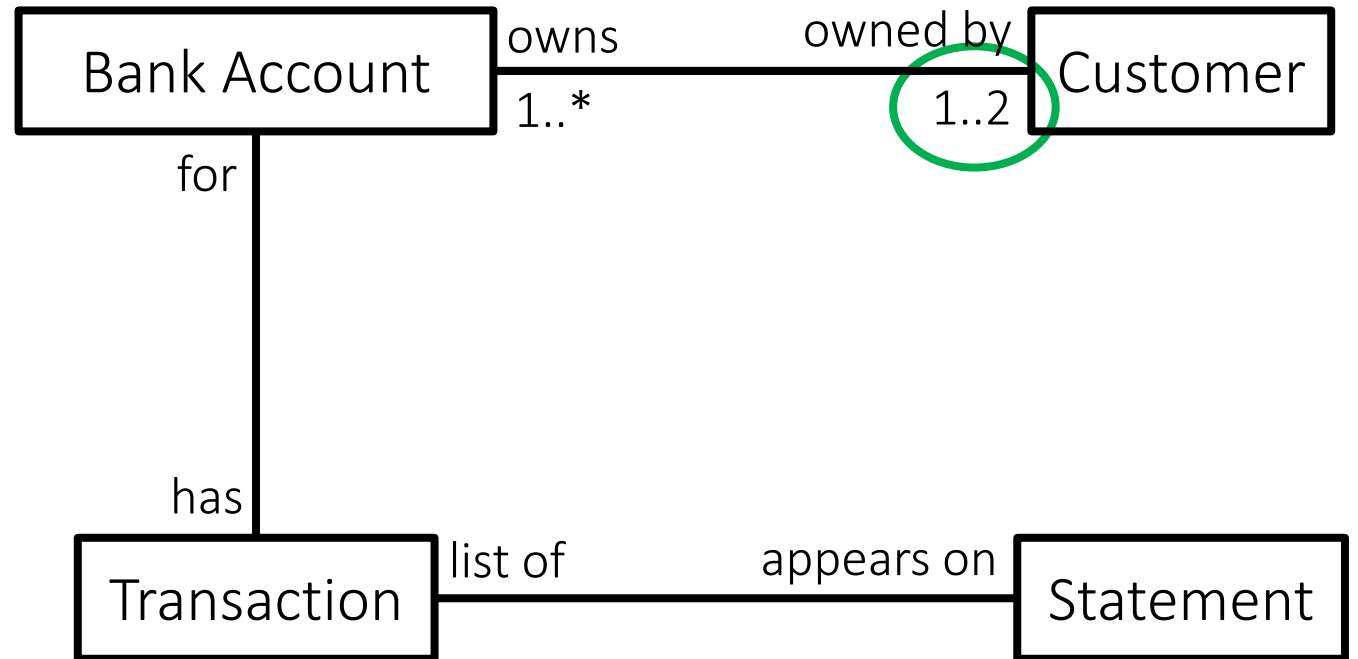
# Multiplicities

How many *Existing Customer(s)* is each *Bank Account* owned by?



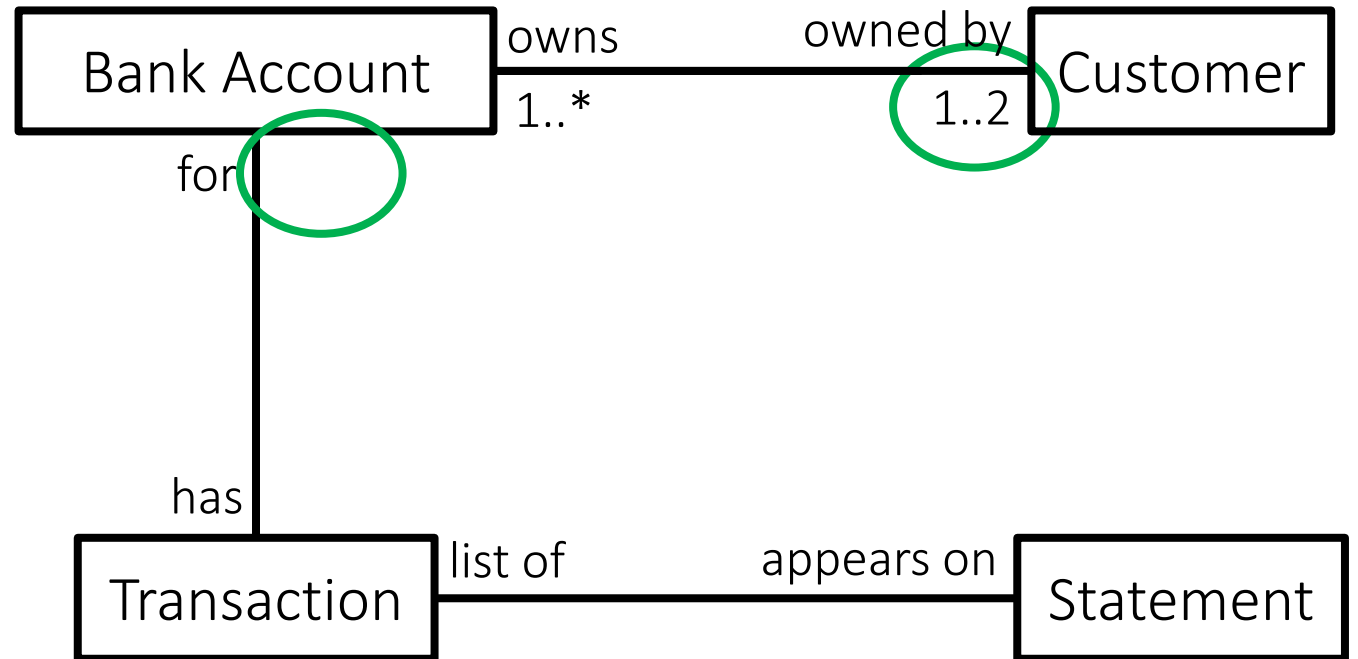
# Multiplicities

How many *Existing Customer(s)* is  
each *Bank Account* owned by?  
1 or 2 (joint accounts)



# Multiplicities

How many *Bank Account*(s) is each *Transaction* for?

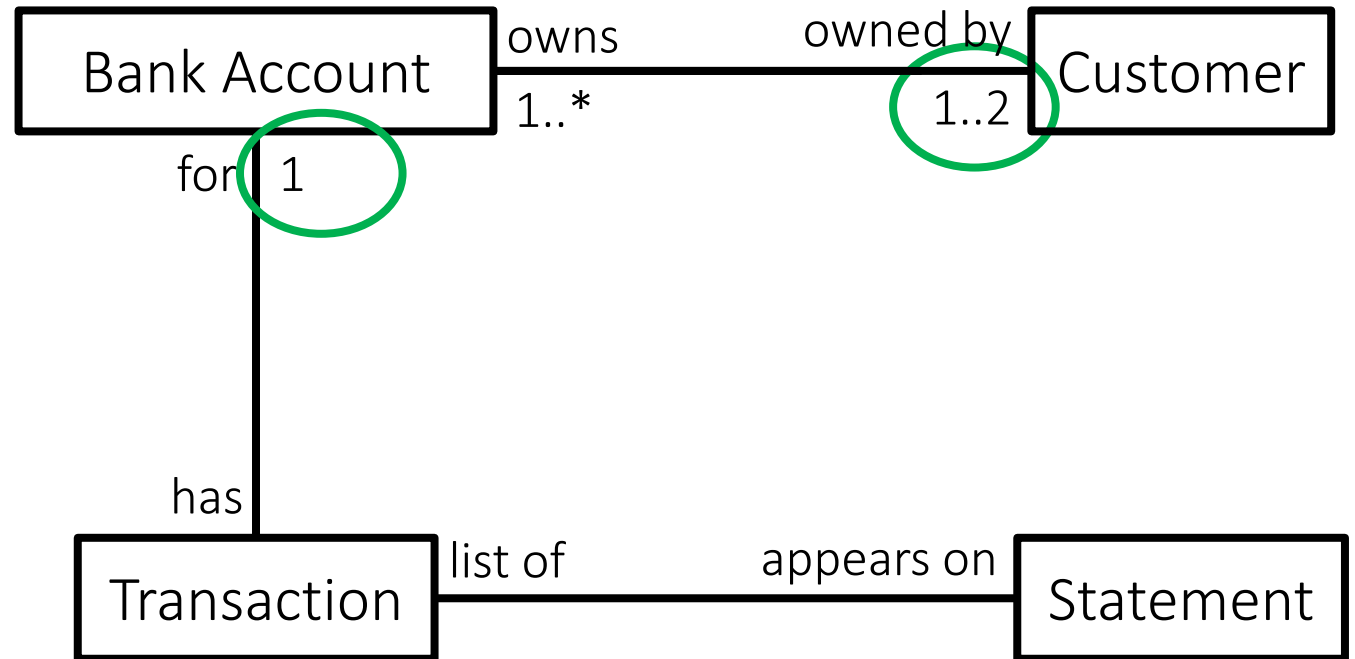


# Multiplicities

How many *Bank Account*(s) is each *Transaction* for?

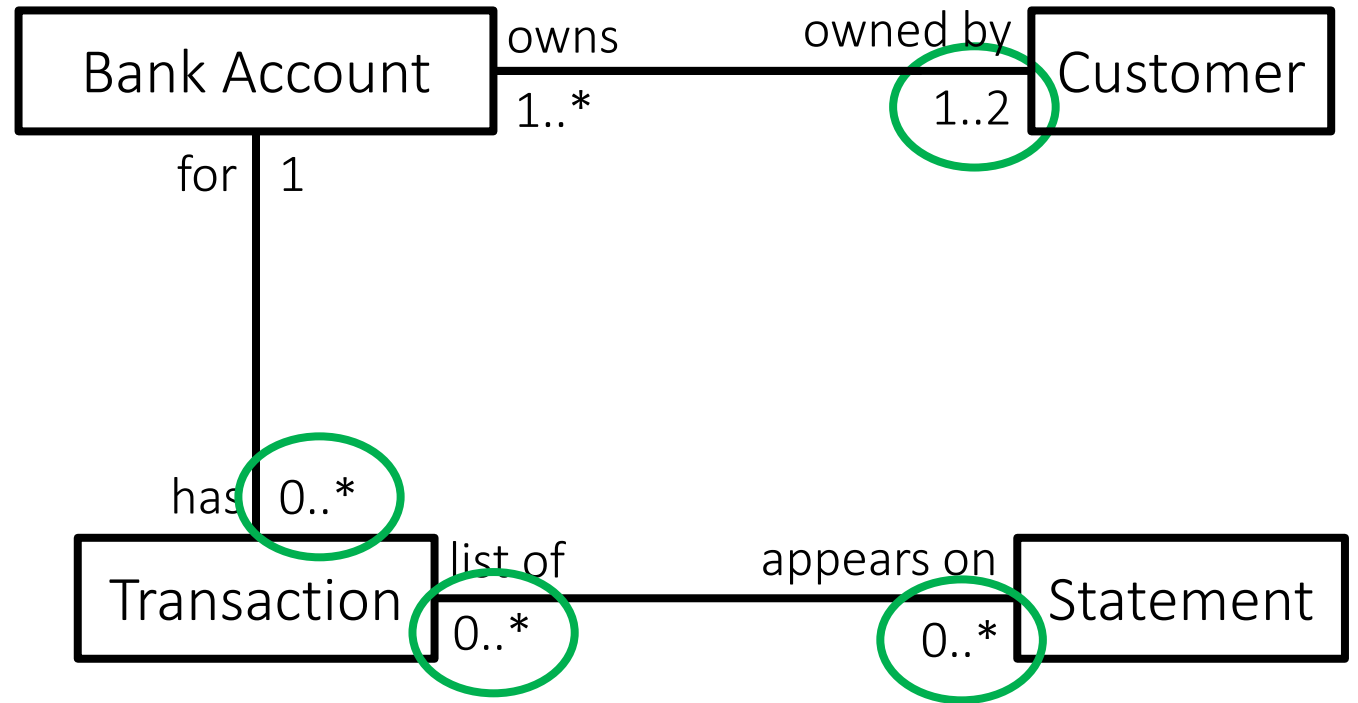
Exactly 1

(Assumption: if a transaction is between two Bank Accounts in the same bank, it's recorded separately for each Bank Account. This way, transactions in the same bank are managed in the same way as transactions to other banks – each transaction will list a type relative to the bank account it is linked to)



# Multiplicities

Keep doing this for both sides of all other associations



# Multiplicities

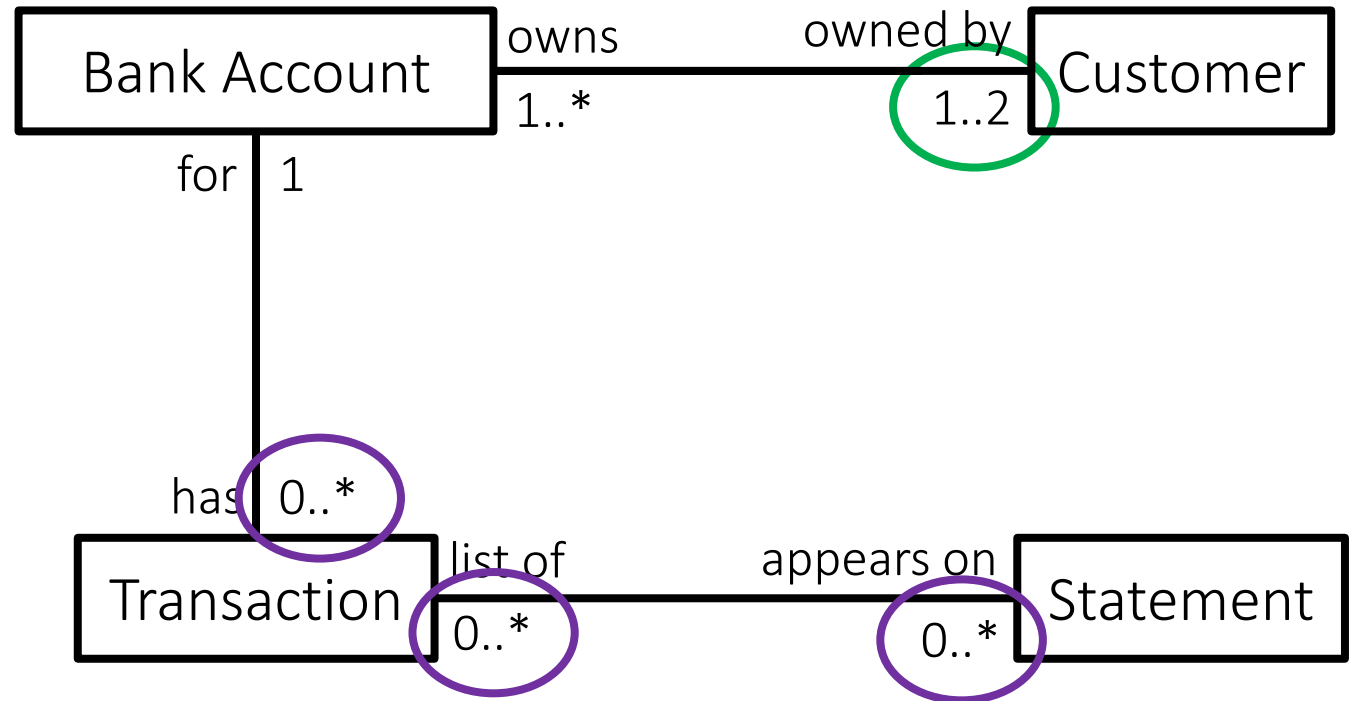
Keep doing this for both sides of all other associations

Association multiplicities must be correct for the system at any point in time

Is it ever possible that a bank account has no transactions?

Is it ever possible that a statement has no transactions?

Is it ever possible that a transaction does not appear on a statement?

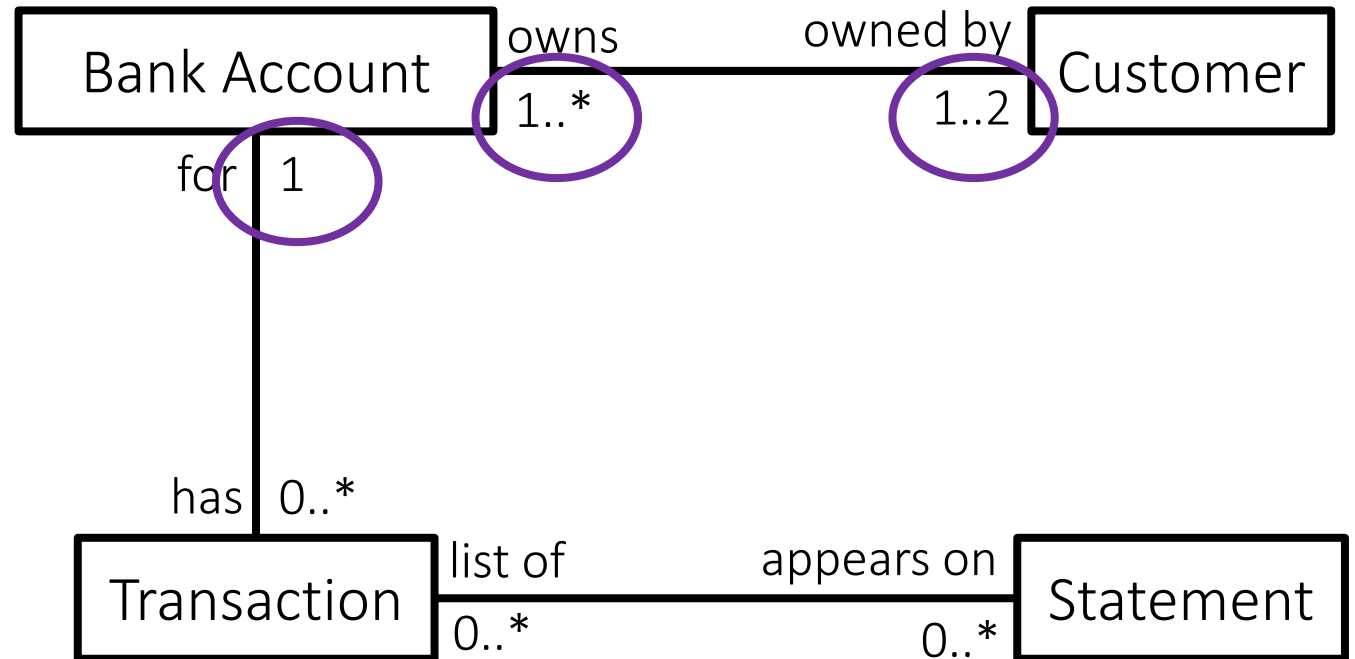


# Multiplicities

Similarly, is it ever possible that an existing customer does not own a bank account?

Is it ever possible for a bank account to not have an owner?

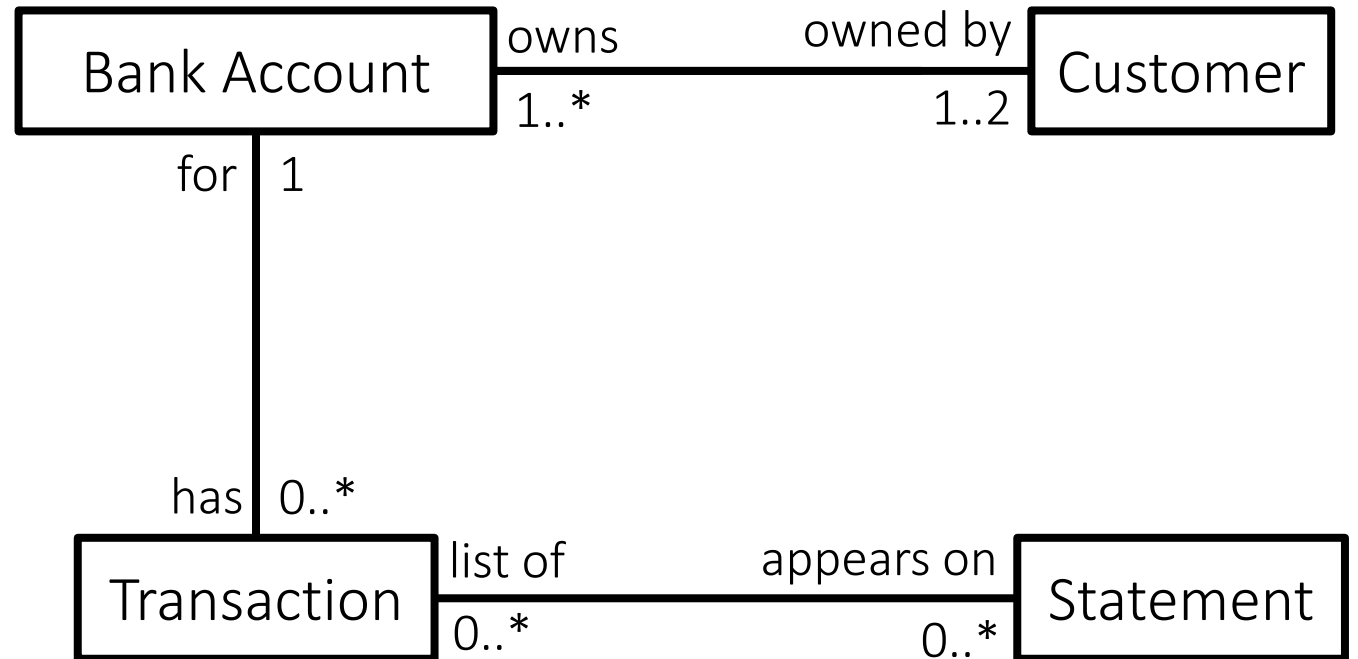
Is it ever possible for a transaction to not be for a bank account?



# Multiplicities

Remember, multiplicities are for each instance of association, to describe how many things an involved class can be related to

They are not for the system as a whole!



# Reading associations

*Each*

1. Class

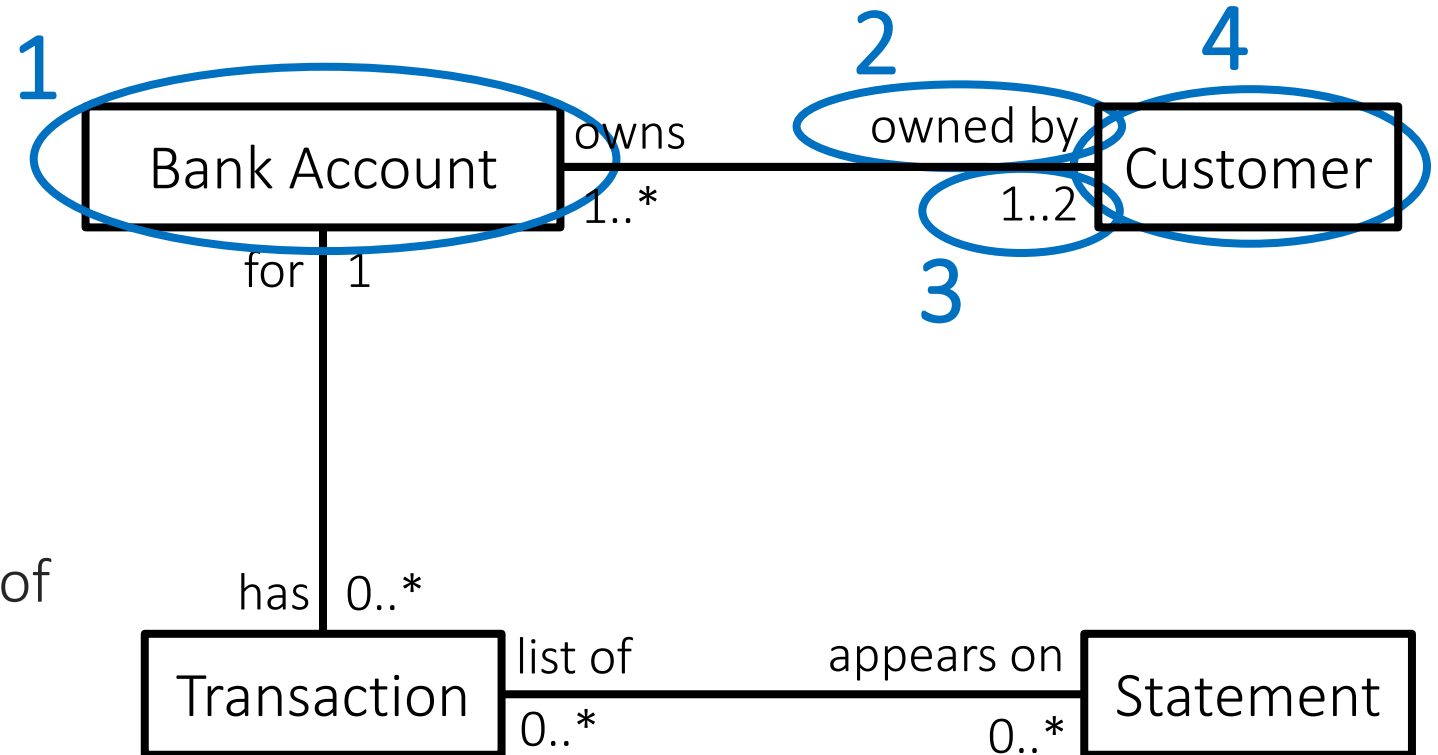
*is*

2. Role on the other end  
(‘destination end’) of the  
association

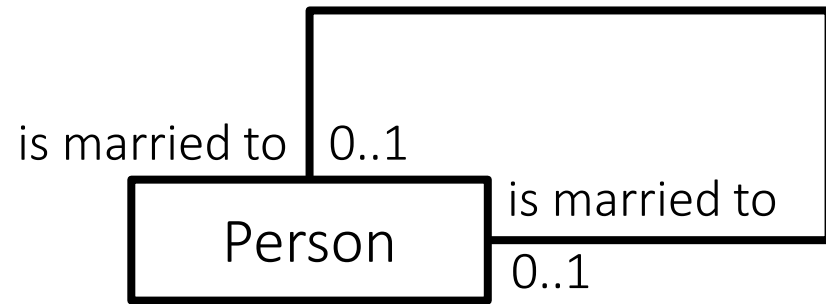
3. Multiplicity on the other end of  
the association

4. Other class

*“Each Bank Account is owned by 1 to  
2 Existing Customer(s)”*



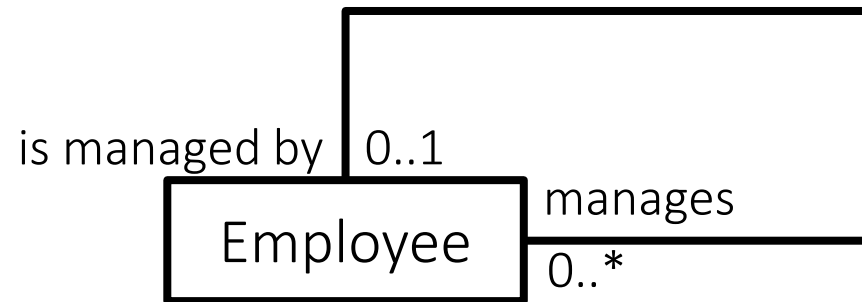
# Reflexive associations



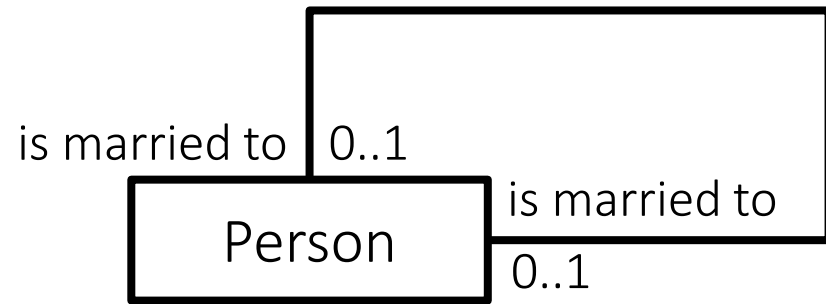
A class can have an association with itself

This is known as a reflexive association

The roles and multiplicities must still make sense

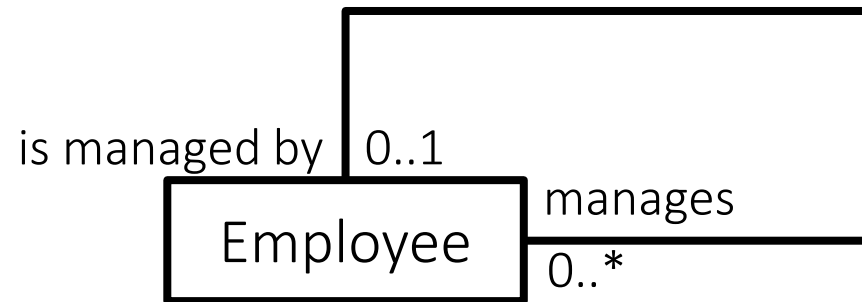


# Reflexive associations

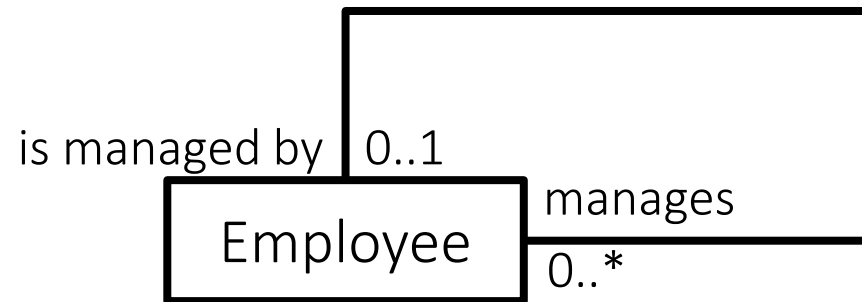
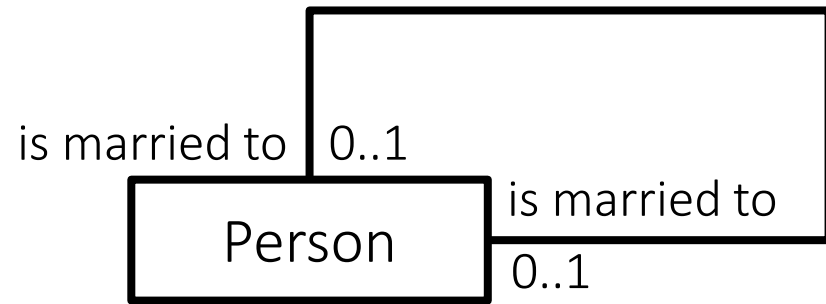


For example, an *Employee* is managed by 0 or 1 *Employee(s)*

If it was an *Employee* is managed by exactly 1 *Employee(s)*, what would be the problem?



# Reflexive associations



For example, an *Employee* is managed by 0 or 1 *Employee(s)*

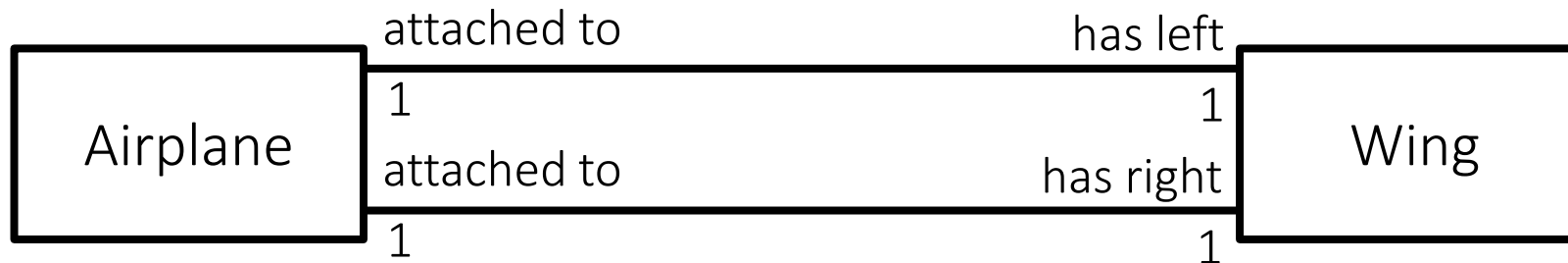
If it was an *Employee* is managed by exactly 1 *Employee(s)*, what would be the problem?

Who manages the boss? The boss's boss? Then who manages the boss's boss? The boss's boss's boss? Uh oh...

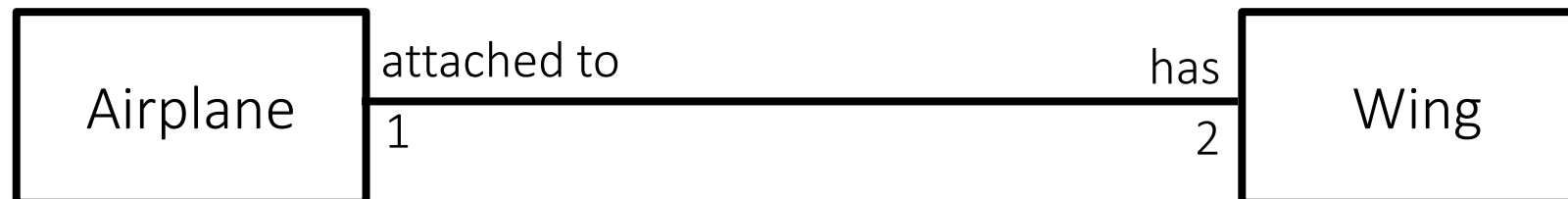
# Multiple associations

It is possible to have more than one association between two classes, as long as the different associations mean different things

This first diagram conveys more precise information:



This second diagram is still correct, but does not specify that there has to be one left wing and one right wing:





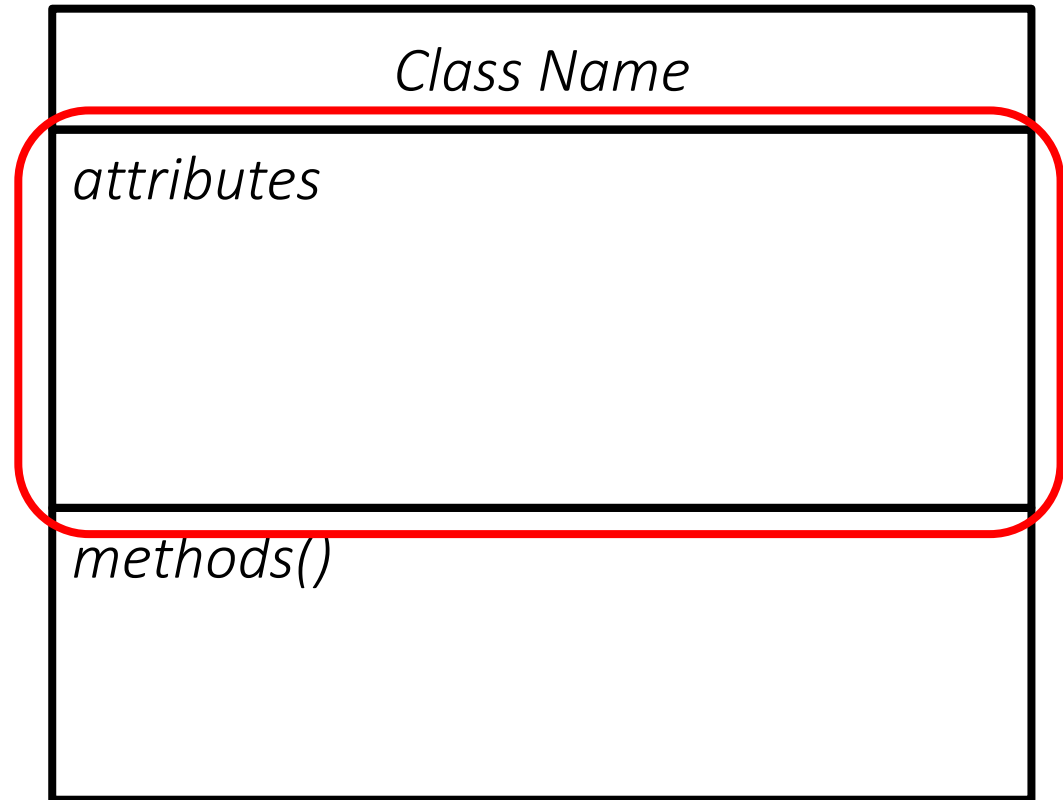
## 2d. Attributes and methods

- a) Attributes
- b) Methods

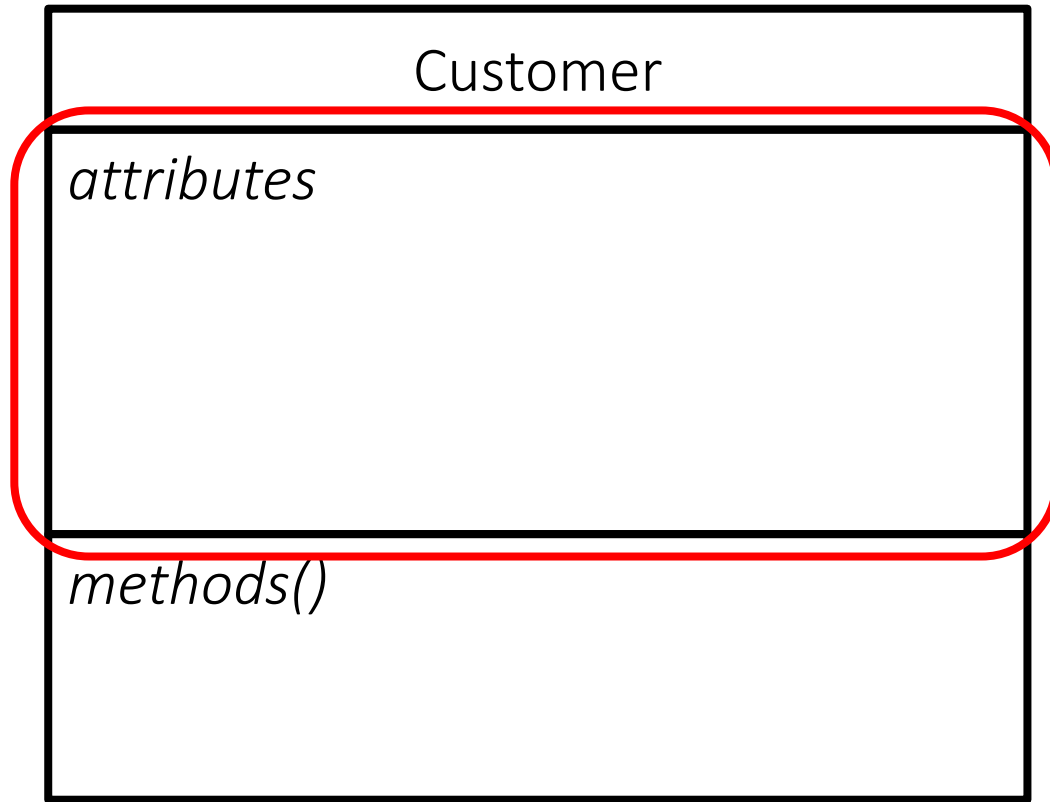
# Attributes

Attributes are types of values that describe objects of a particular class

Attribute names start with a lower case letter, to distinguish them from class names

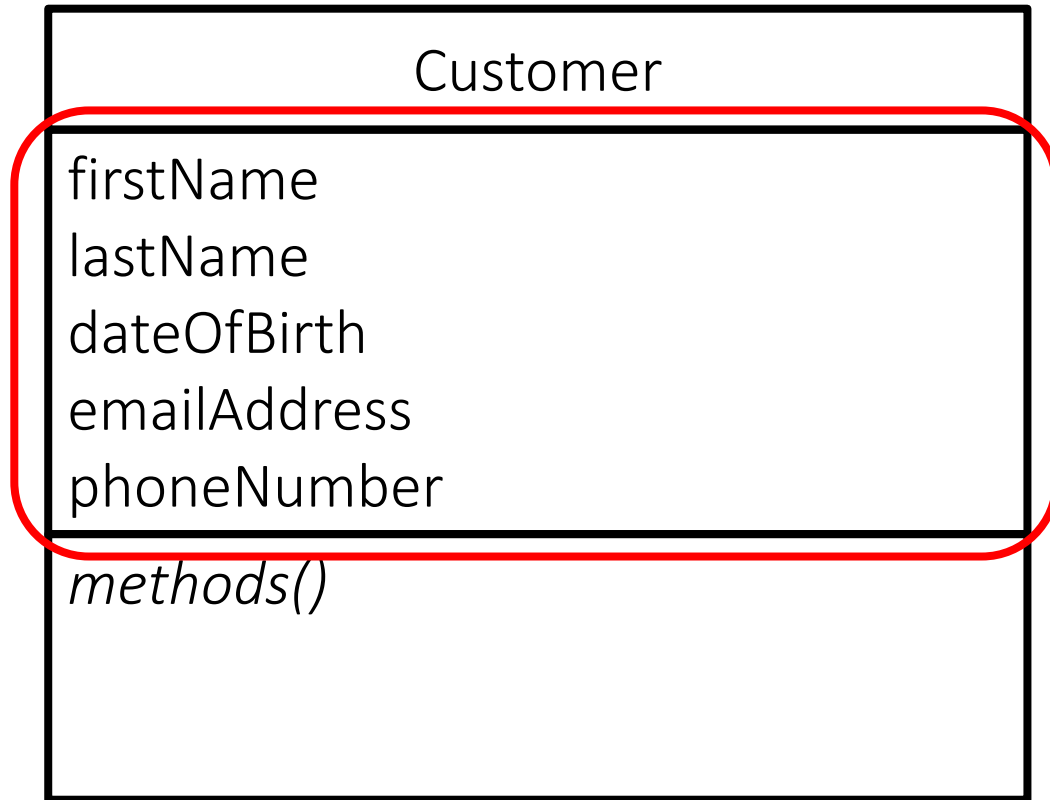


# Attributes



For the customer class in the bank account management system, what might be some attributes?

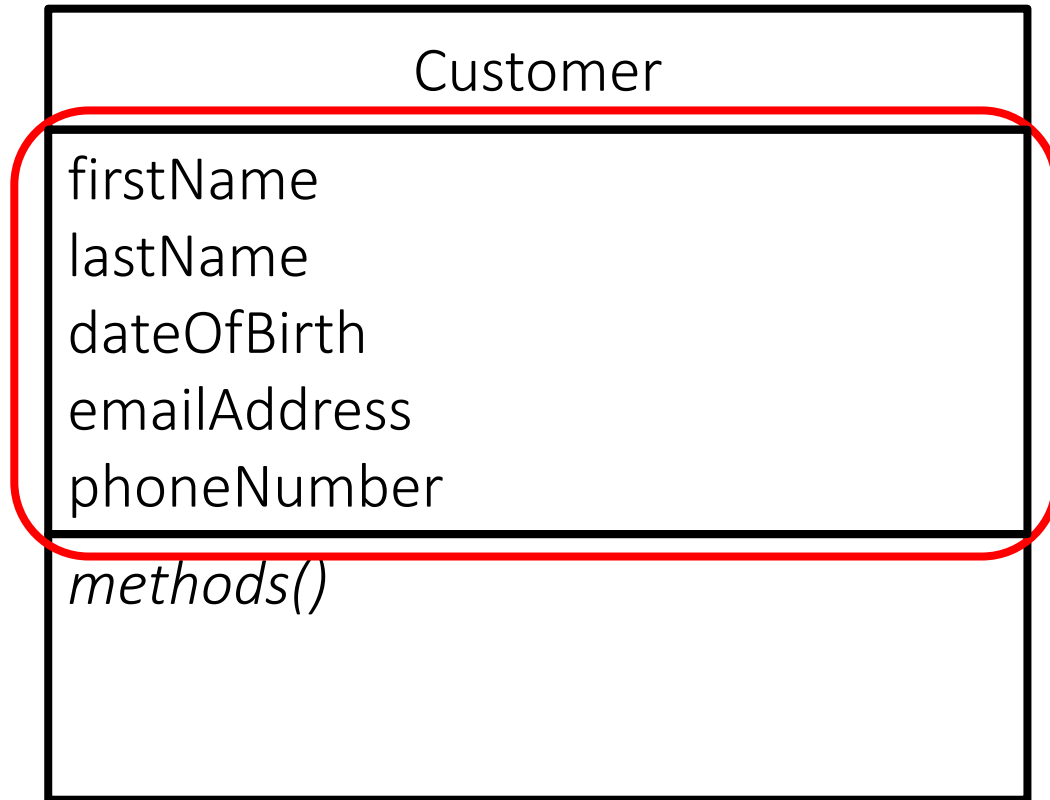
# Attributes



For the customer class in the bank account management system, what might be some attributes?

These might look pretty similar to things from the data dictionary...

# Attributes

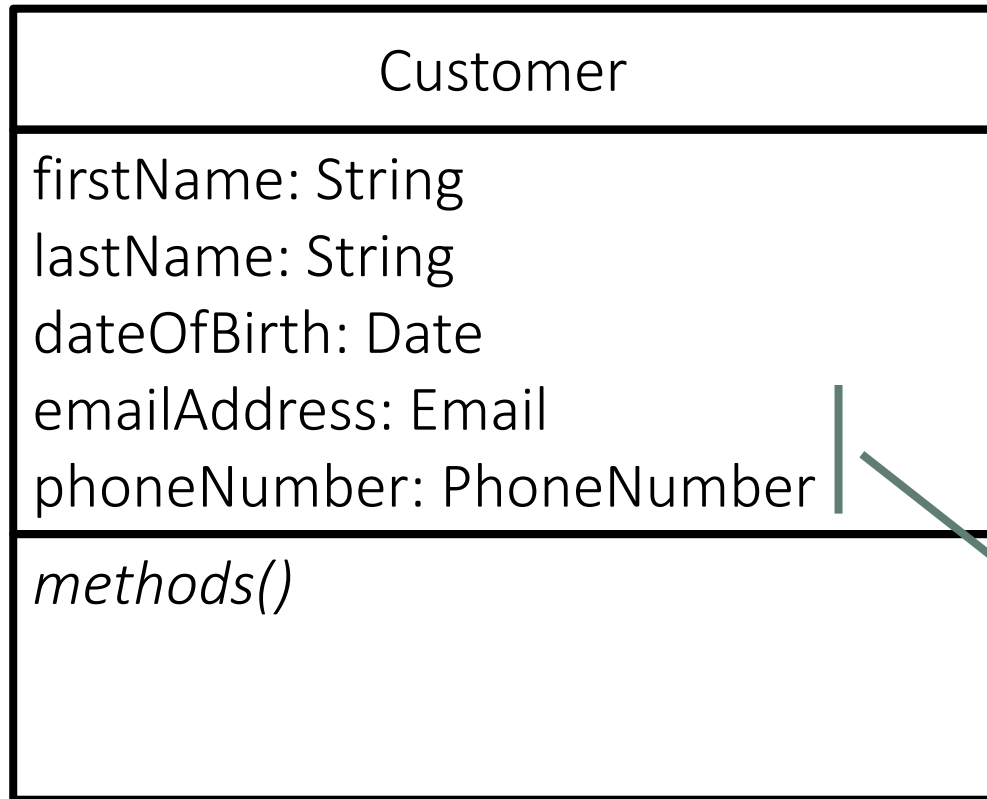


For the customer class in the bank account management system, what might be some attributes?

These might look pretty similar to some things from the data dictionary...

Things like the customer's accounts will be other classes that the Customer class is associated with

# Attributes



Attributes are further defined by adding a data type

String, Integer, Date, etc...

Again, these might look pretty similar to some things from the data dictionary...

The data types Email and PhoneNumber would have to be further defined to provide additional clarity, similar to a data dictionary, although UML uses a different way of formally defining complex data types that we won't cover in this unit

# Attributes

| Customer                                                                                                      |
|---------------------------------------------------------------------------------------------------------------|
| firstName: String<br>lastName: String<br>dateOfBirth: Date<br>emailAddress: Email<br>phoneNumber: PhoneNumber |
| <i>methods()</i>                                                                                              |

Further details such as visibility, multiplicity, default values, and other properties are also possible, but these won't be covered in the lecture or expected in assessment

These often are more important at the design stage rather than analysis

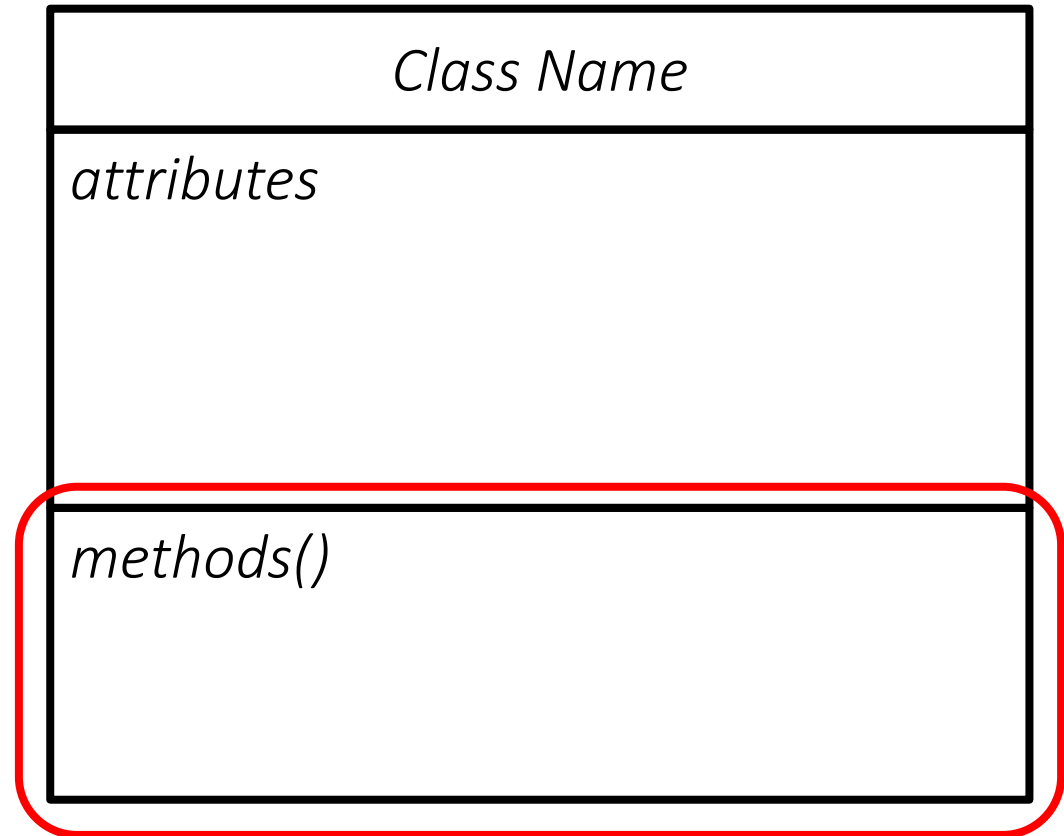
Fully described in the textbook

# Methods

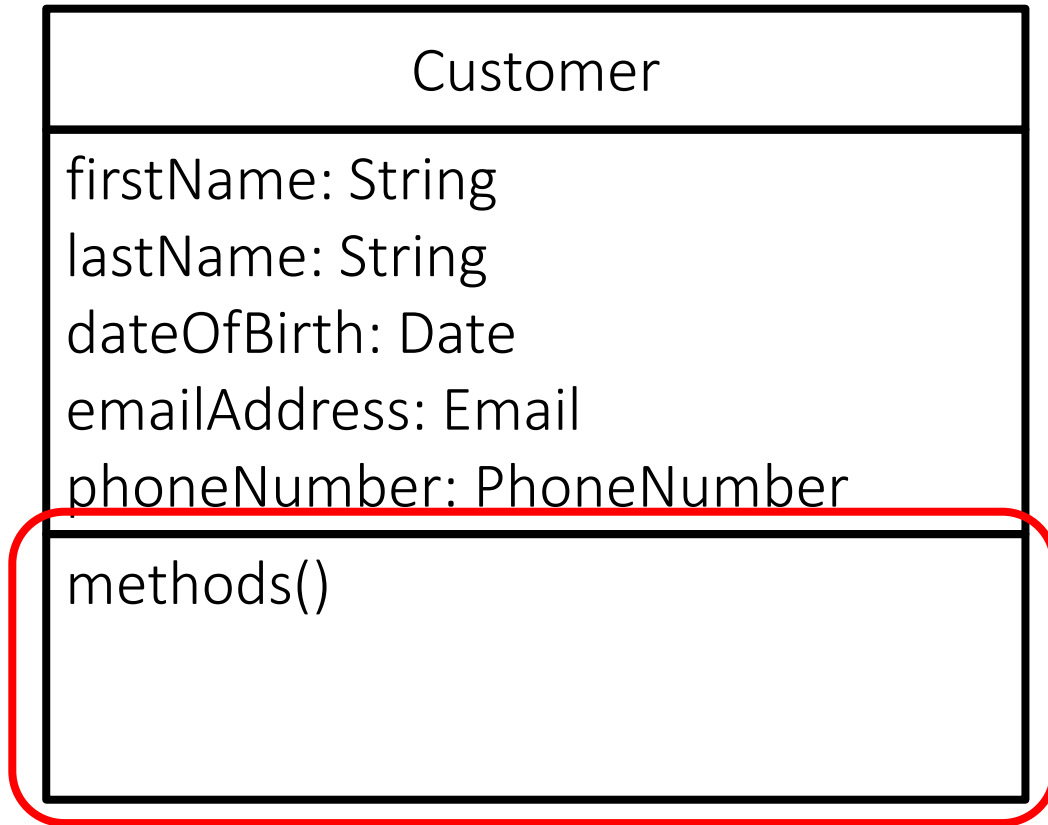
Methods describe behaviours of objects – what the object does, or what can be done to it

Instances of objects carry out the methods listed

Method names start with lower case letters and end with parenthesis to distinguish them from class names and attribute names

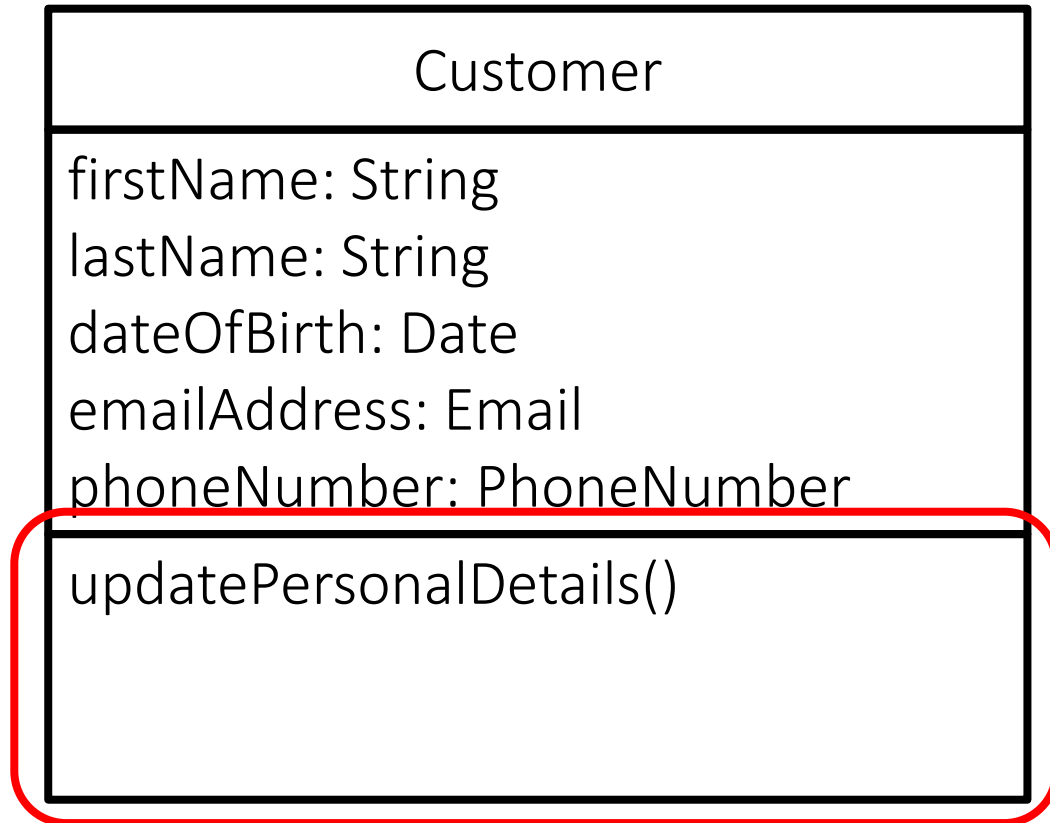


# Methods



For the customer class in the bank account management system, what might be some methods?

# Methods

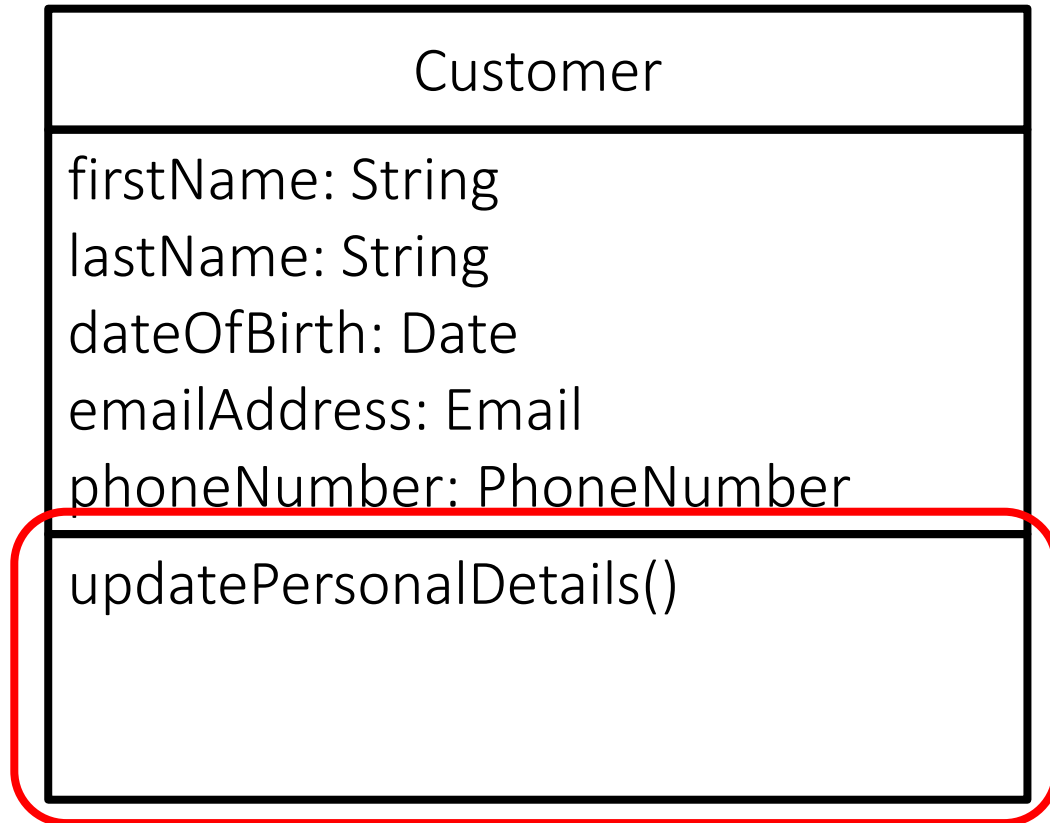


For the customer class in the bank account management system, what might be some methods?

These might look pretty similar to some things from the event table...

But what about opening a new account, updating account information, withdrawals, deposits, etc?

# Methods



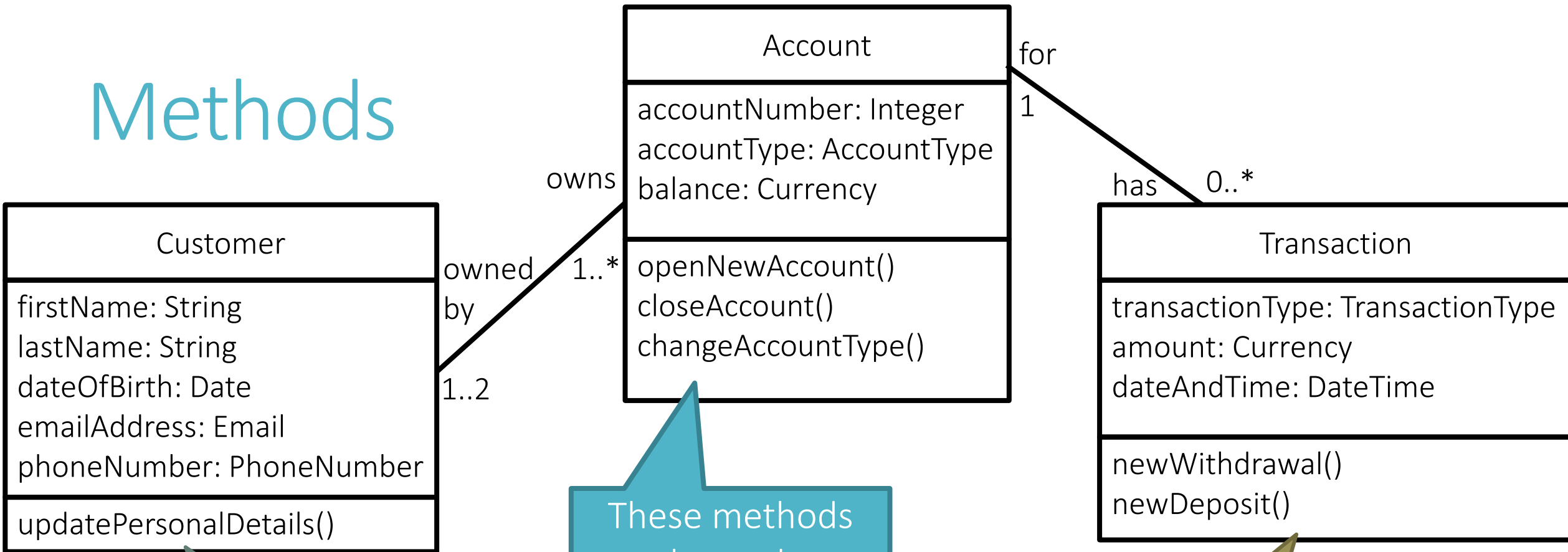
For the customer class in the bank account management system, what might be some methods?

These might look pretty similar to some things from the event table...

But what about opening a new account, updating account information, withdrawals, deposits, etc?

These are more likely to be methods for other classes, such as the Account class or Transaction class

# Methods



This method works mostly on attributes of the Customer class

These methods work mostly on attributes of the Account class

These methods work mostly on attributes of the Transaction class



## 4. Advanced concepts

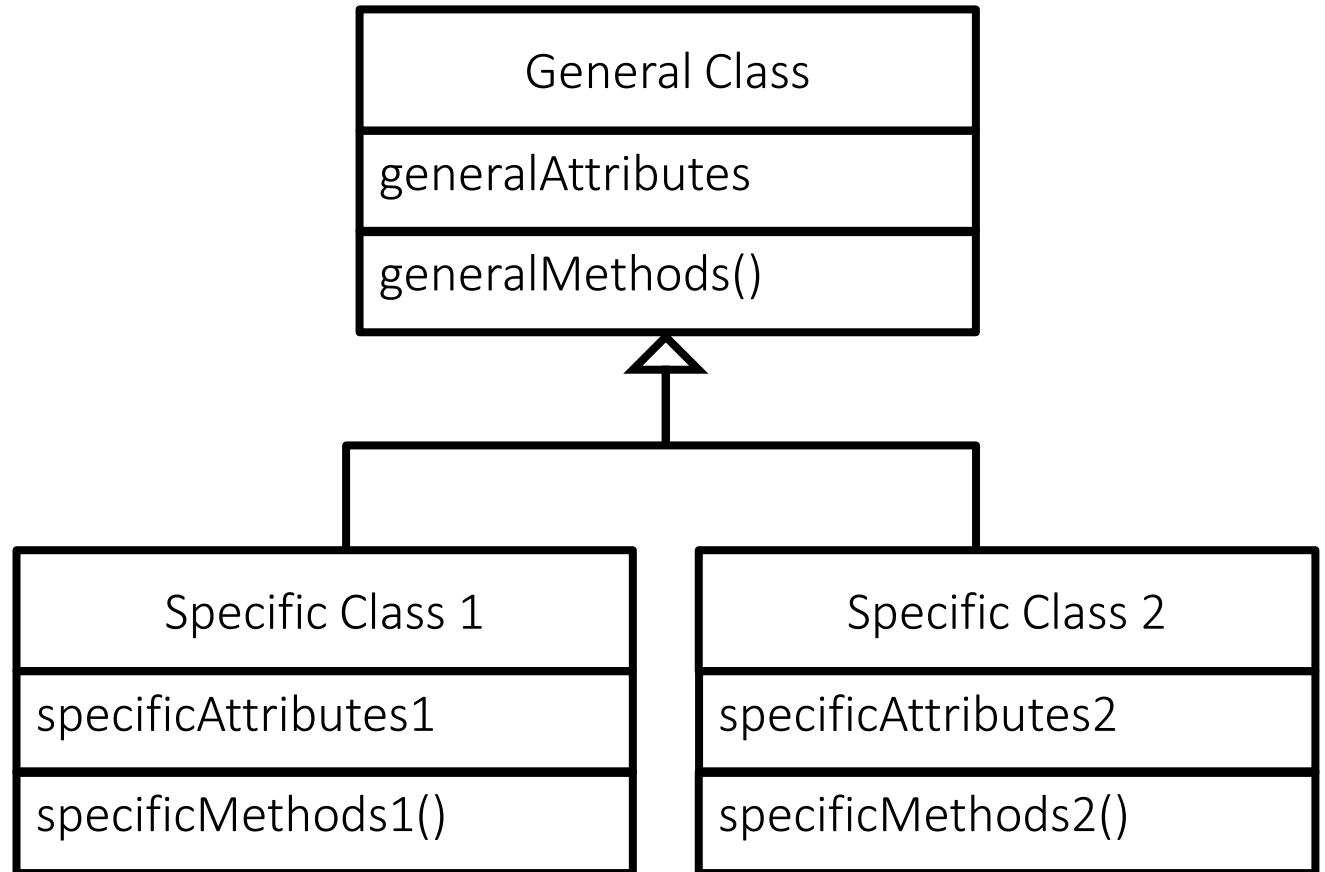
- a) Generalisation
- b) Abstract classes
- c) Association classes

# Generalisation in class diagrams

Similar to generalisation in use case diagrams

Generalisation in class diagrams is used between classes

A specialised class inherits description (attributes) and behaviour (methods) from the general class



# Generalisation in class diagrams

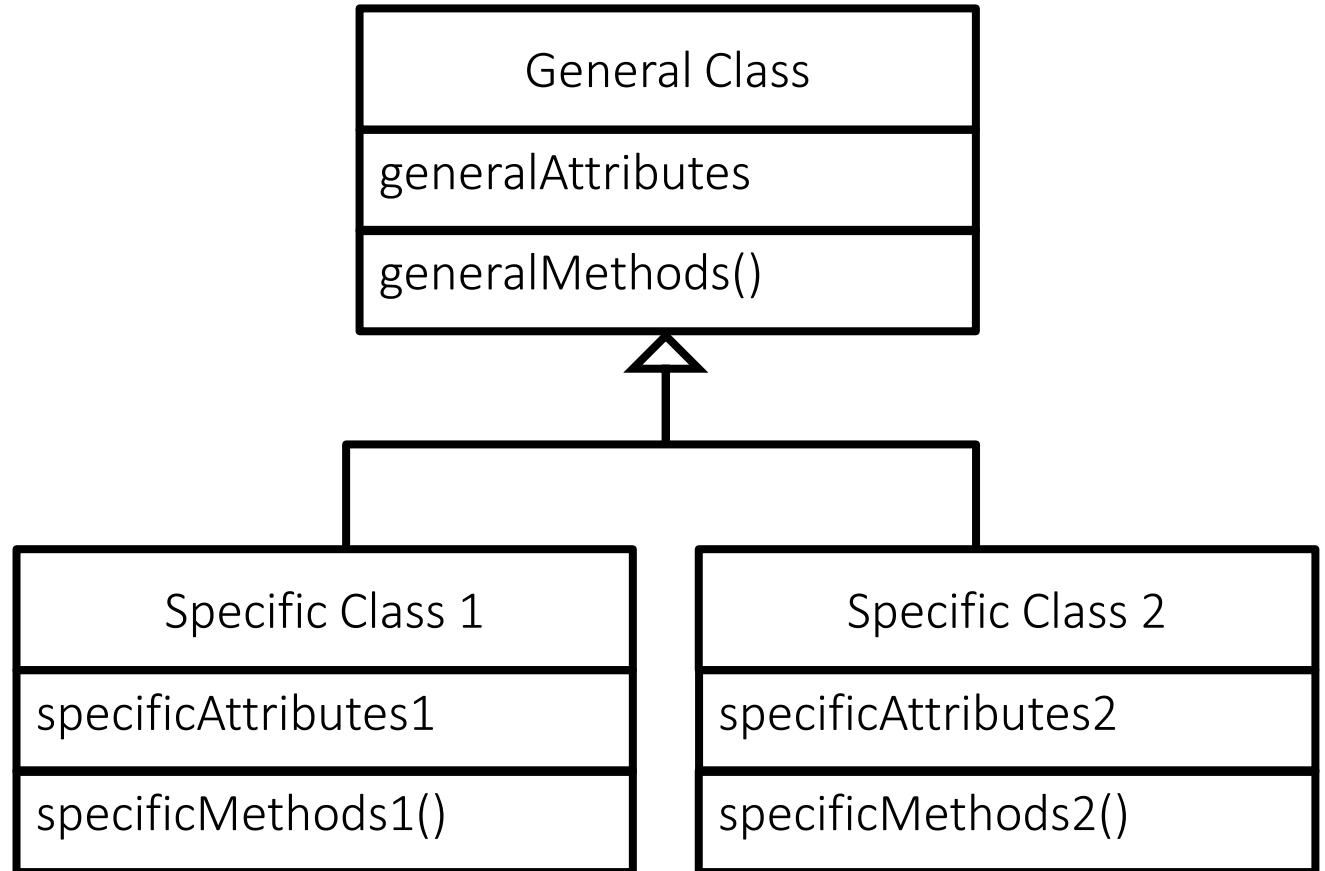
So, Specific Class 1 has:

generalAttributes

specificAttributes1

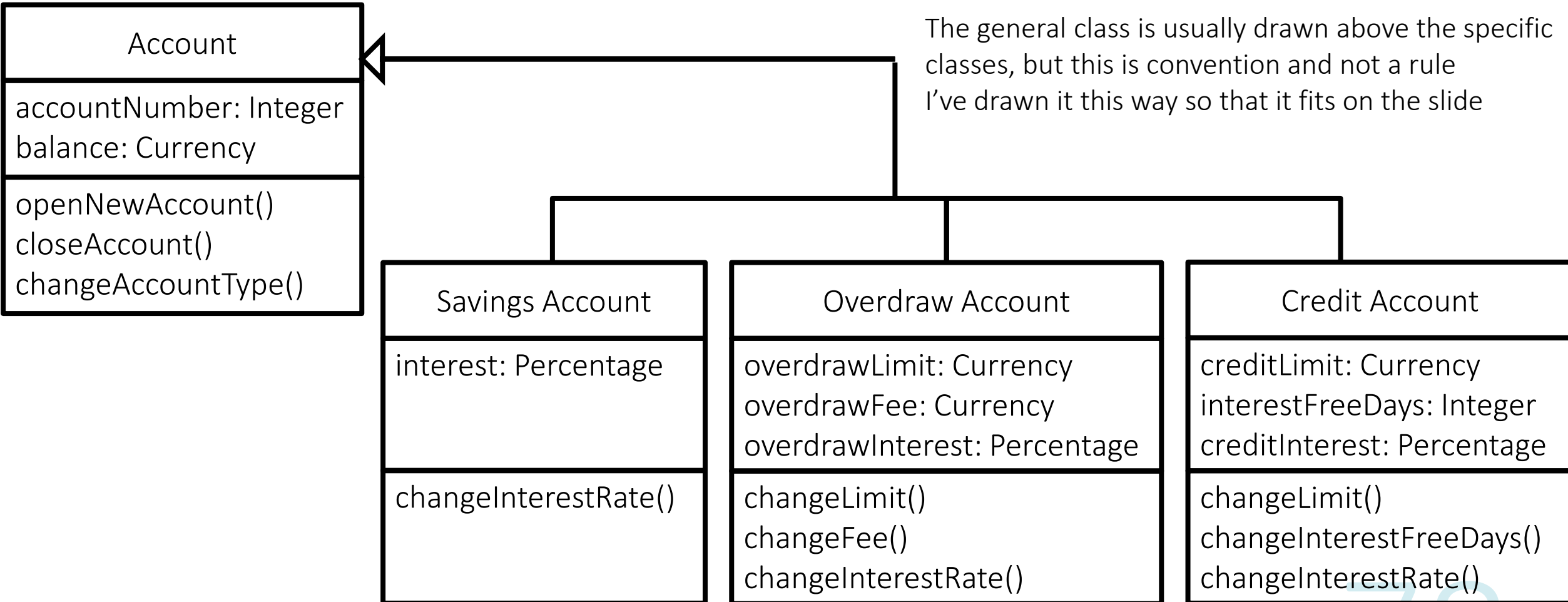
generalMethods()

specificMethods1()

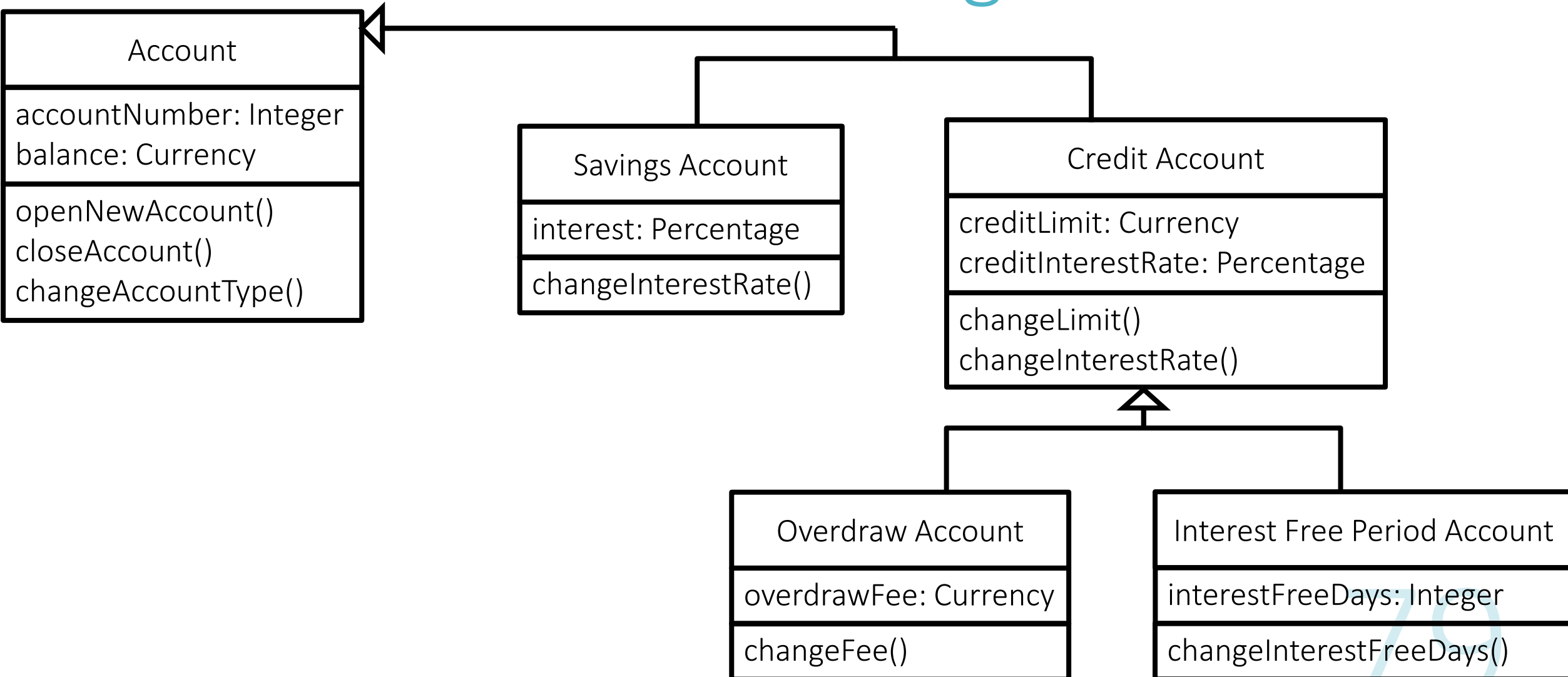


# Generalisation in class diagrams

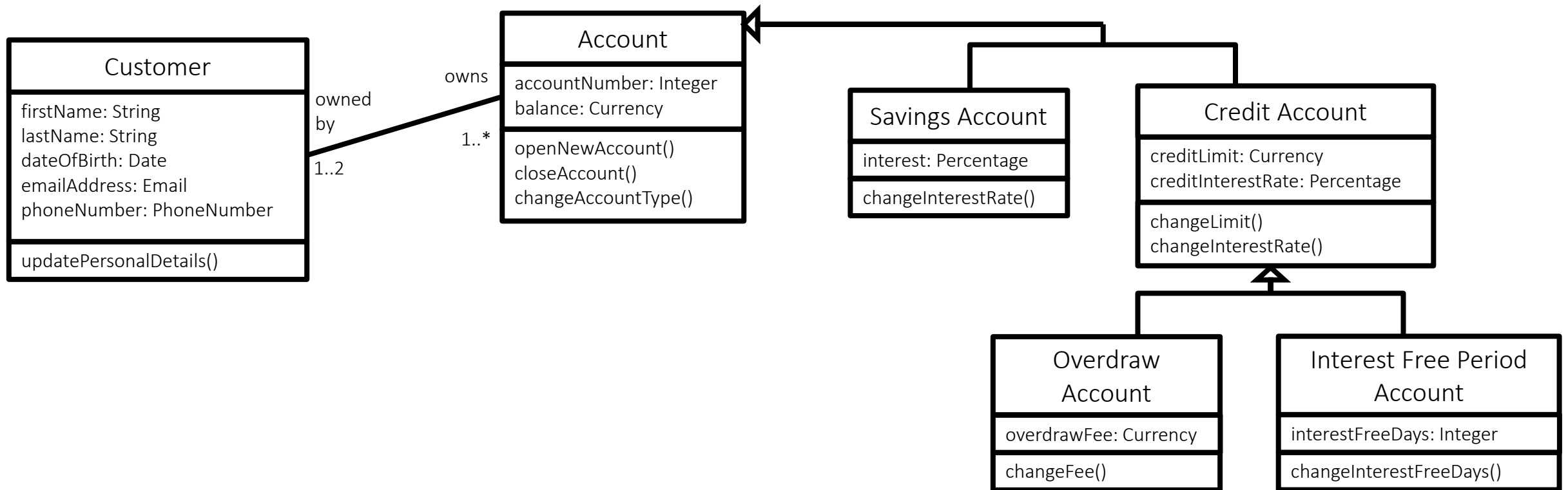
The general class is usually drawn above the specific classes, but this is convention and not a rule I've drawn it this way so that it fits on the slide



# Generalisation in class diagrams



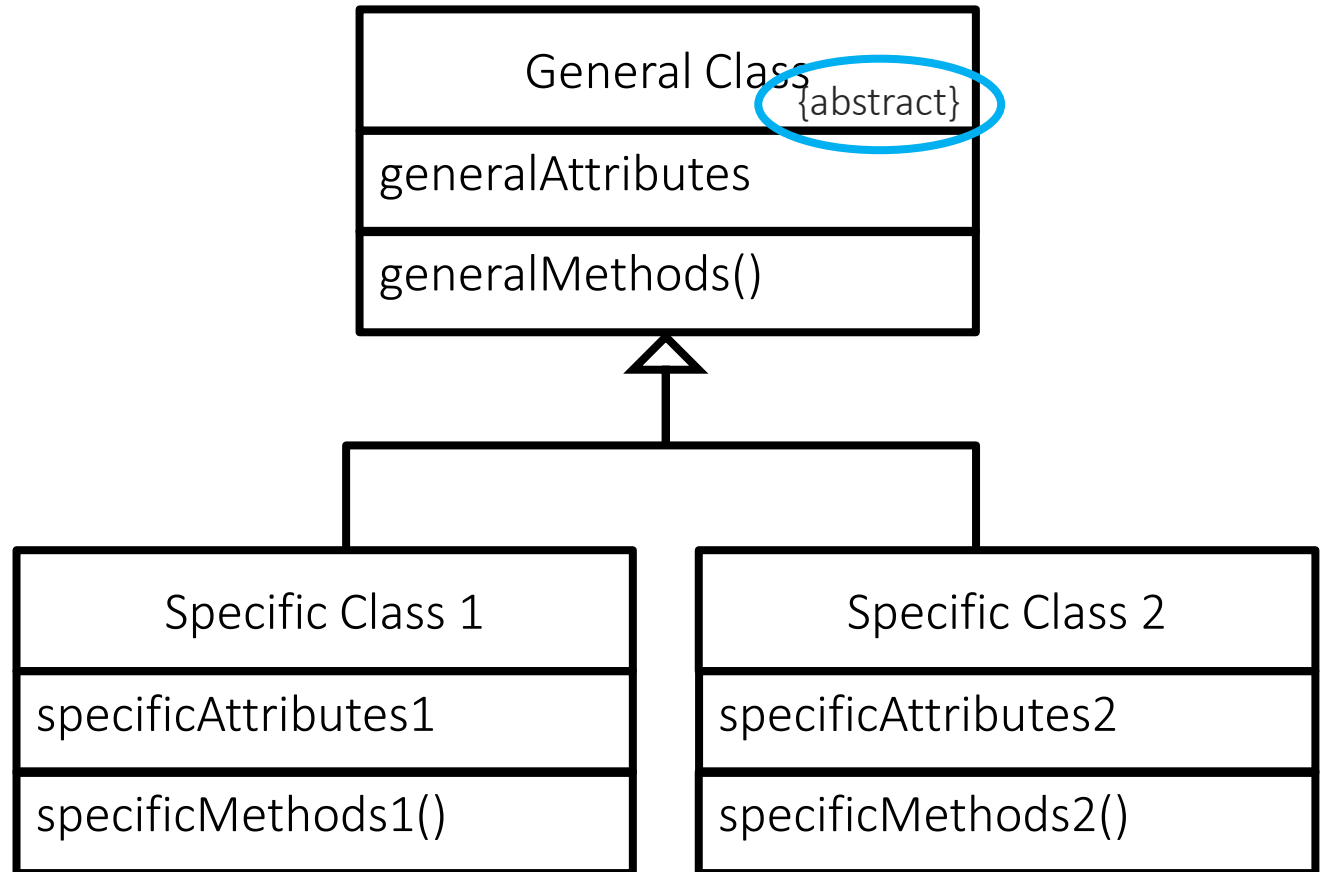
# Generalisation in class diagrams



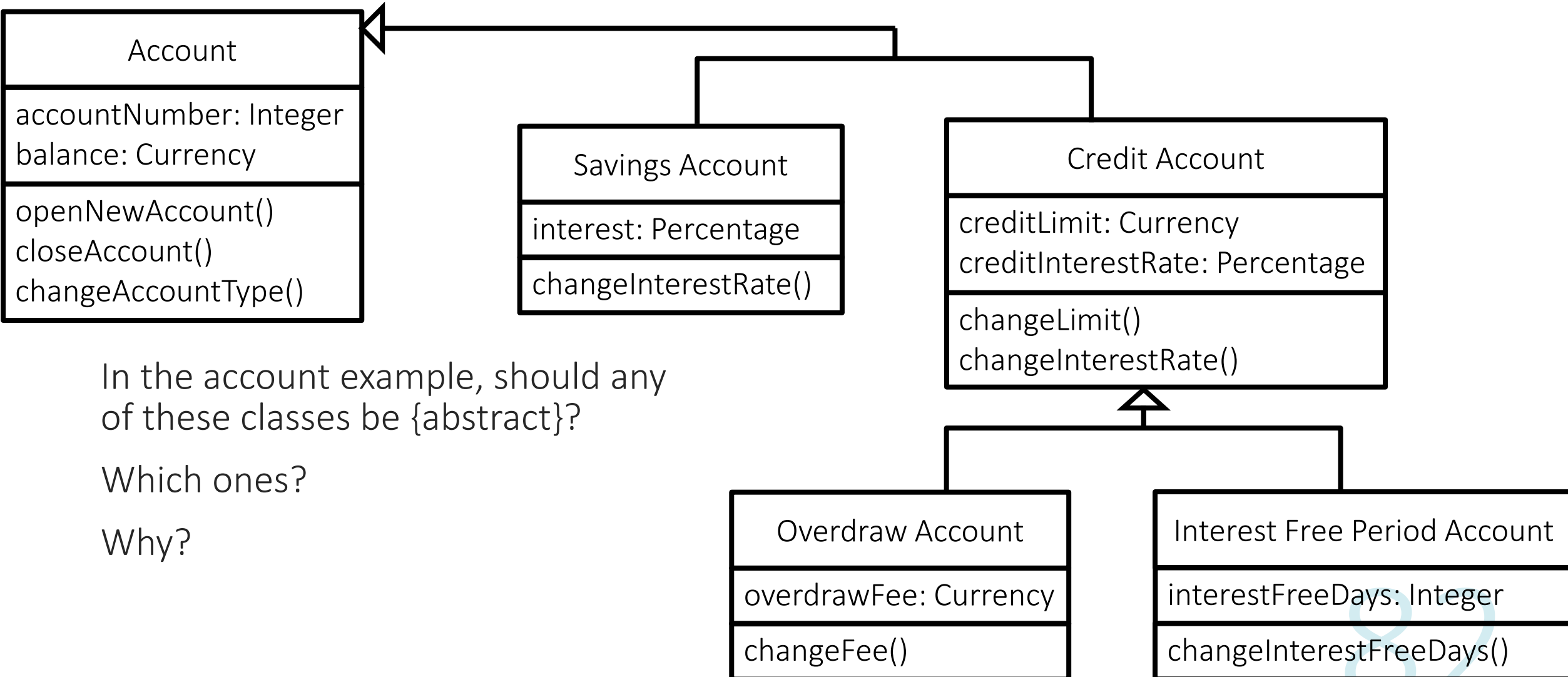
# Abstract classes

If you don't want there to be able to be instances (objects) of a general class, then make that class {abstract}

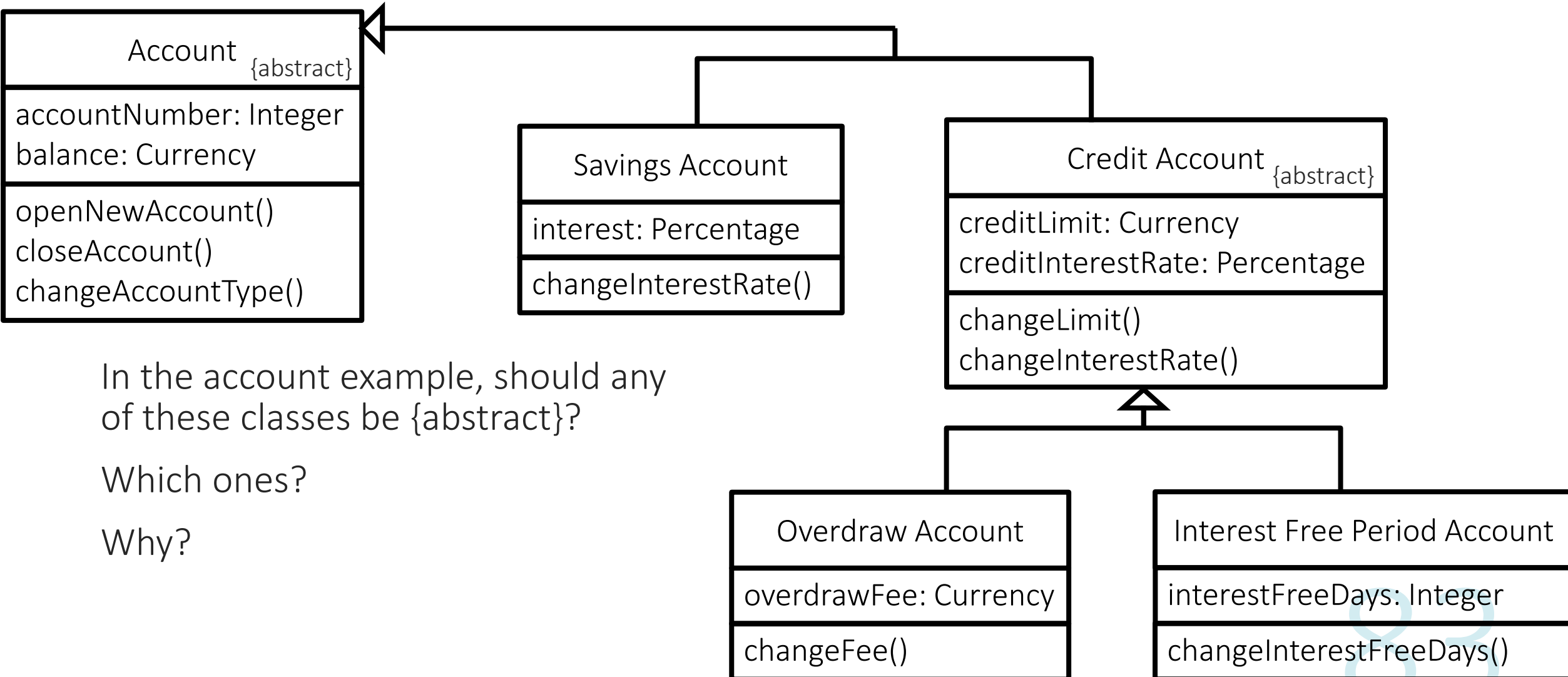
This means there can only be objects of type *Specific Class 1* or *Specific Class 2*



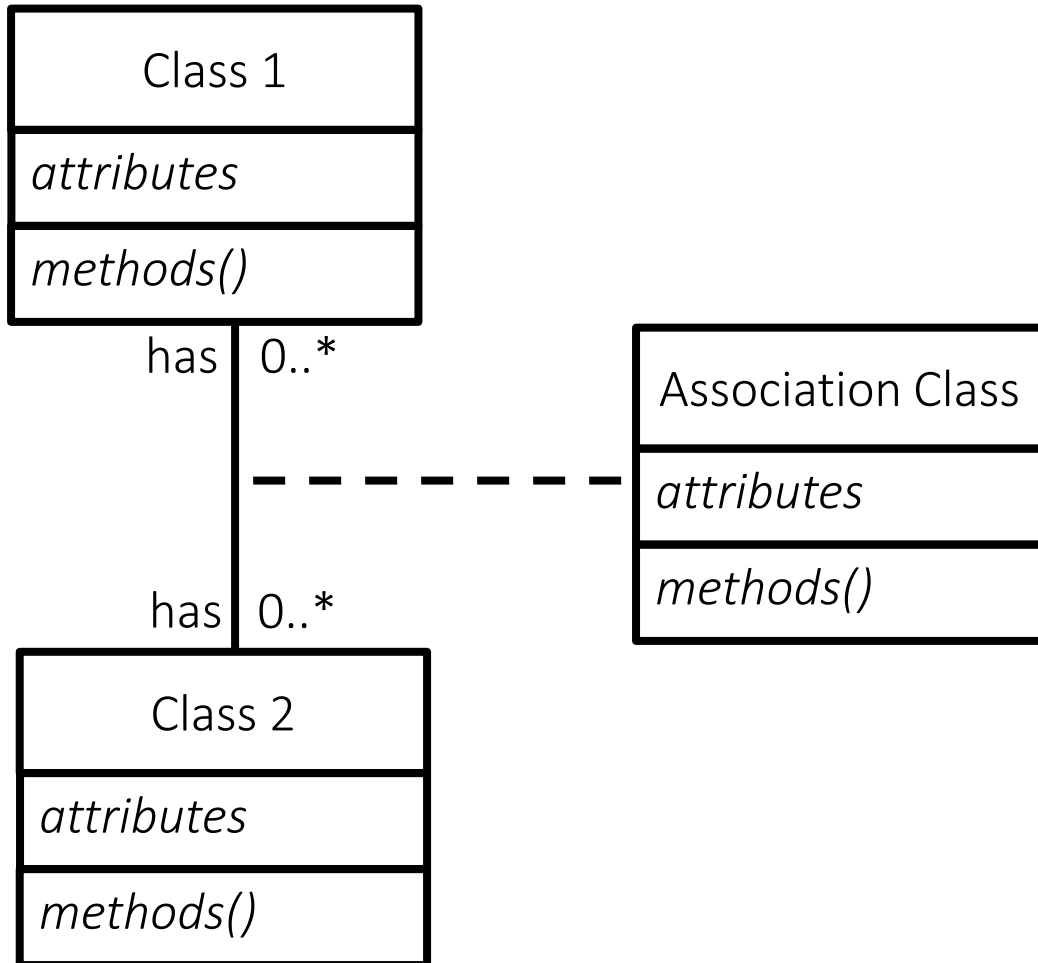
# Abstract classes



# Abstract classes



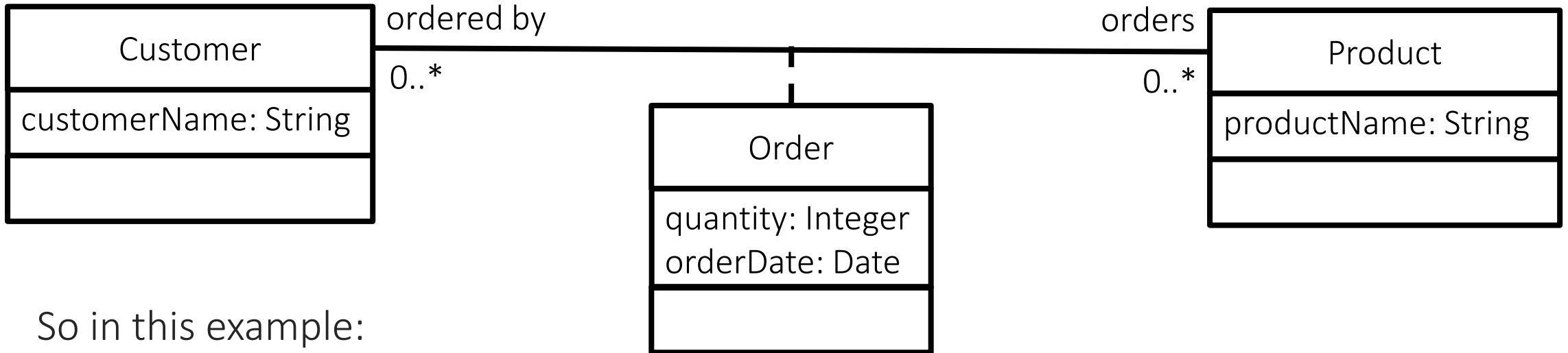
# Association classes



Association classes are used when an association should have attributes or operations of its own

For each pair of associated objects of the associated classes, there exists an association object of the association class

# Association classes

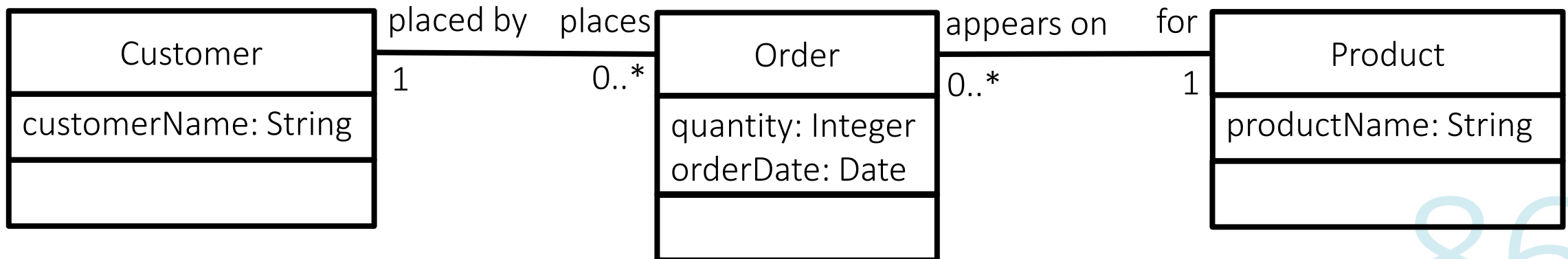
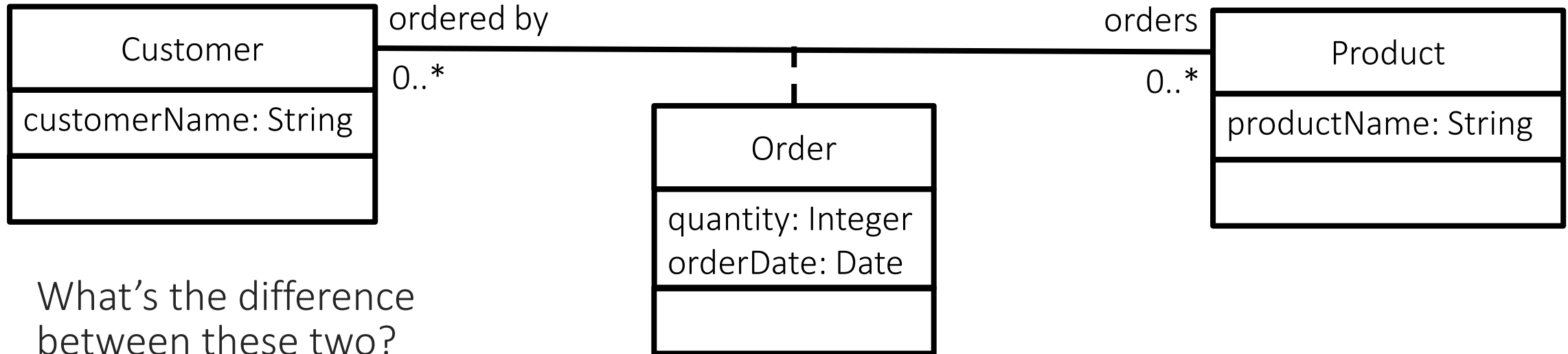


So in this example:

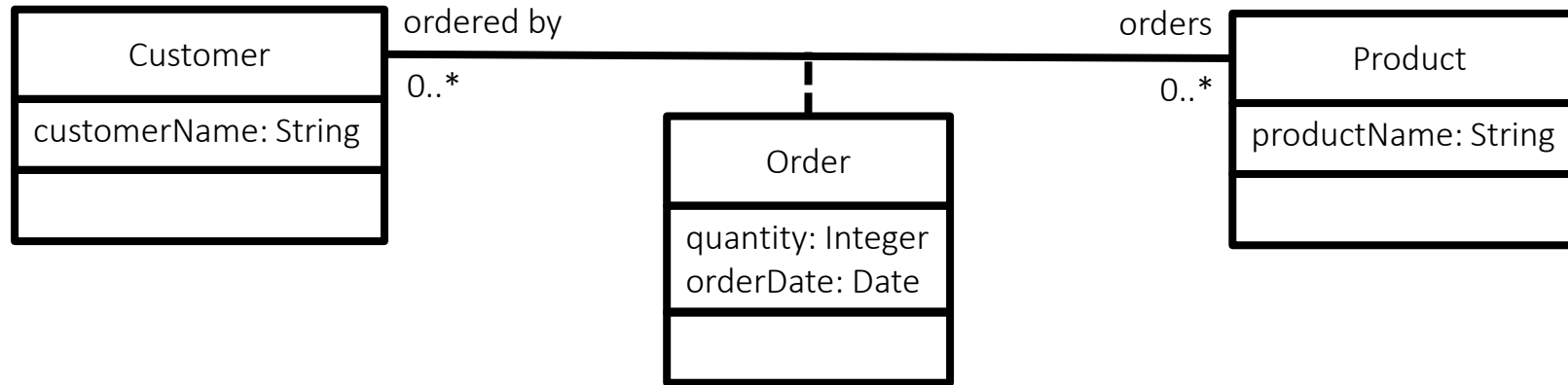
Each customer-product pair results in an order, and each order is for a specific customer-product pair

In UML class diagrams, each pair of objects can only participate in a given association once, so for each customer-product pair there is only one ordered by/orders association, and one resulting order

# Association classes



# Association classes

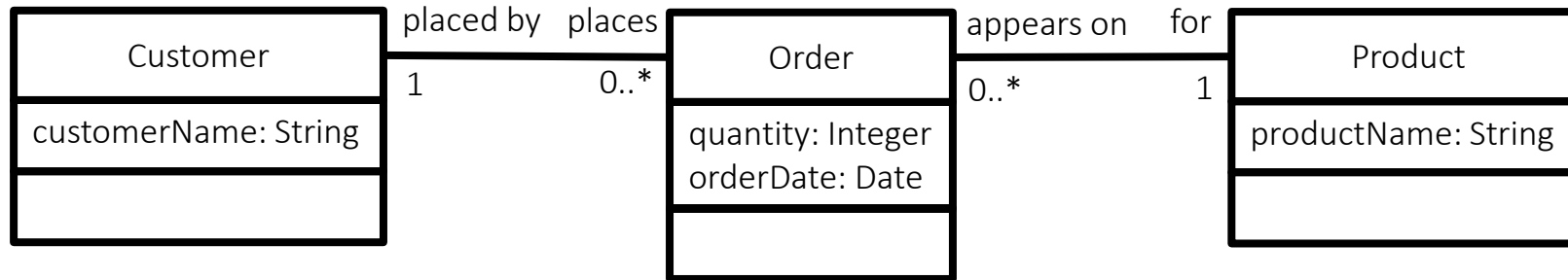


Using the association class enforces that for each and every customer-product pair, there is a different order

Note that using the association class means that a particular customer can't have more than one order for a particular product...

| Customer | Order                     | Product             |
|----------|---------------------------|---------------------|
| Alice    | Quantity 2, on 13/09/2019 | iPhone X            |
| Alice    | Quantity 1, on 20/09/2019 | Samsung Galaxy Fold |
| Bob      | Quantity 4, on 20/09/2019 | iPhone X            |

# Association classes

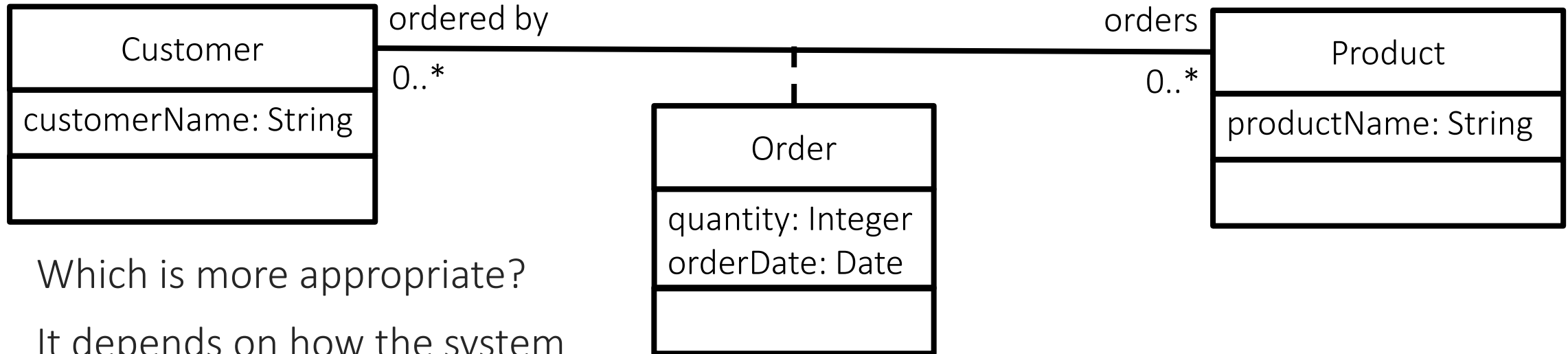


In this version, each Customer can place multiple Orders, each Order is for a particular Product

This means that a particular Customer could place multiple Orders, but those Orders are for the same Product (essentially allowing the Customer to order the same Product multiple times)

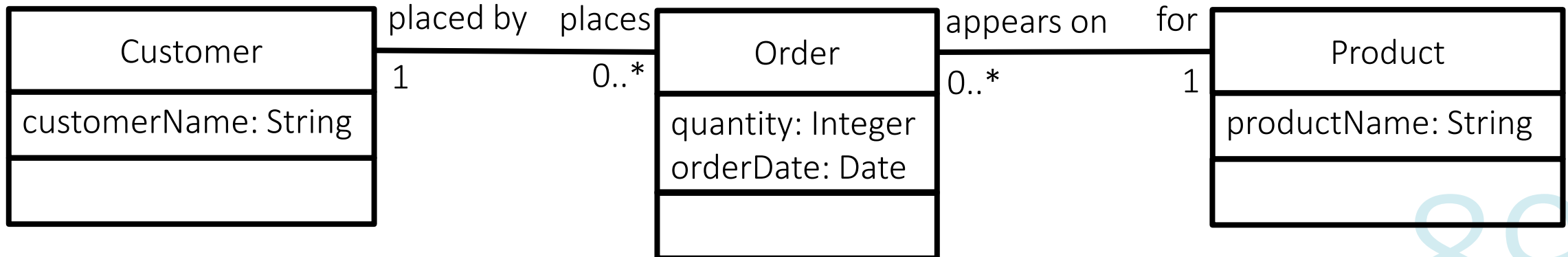
| Customer | Order                     | Product  |
|----------|---------------------------|----------|
| Alice    | Quantity 2, on 13/09/2019 | iPhone X |
| Alice    | Quantity 1, on 20/09/2019 | iPhone X |
| Alice    | Quantity 4, on 27/09/2019 | iPhone X |

# Association classes

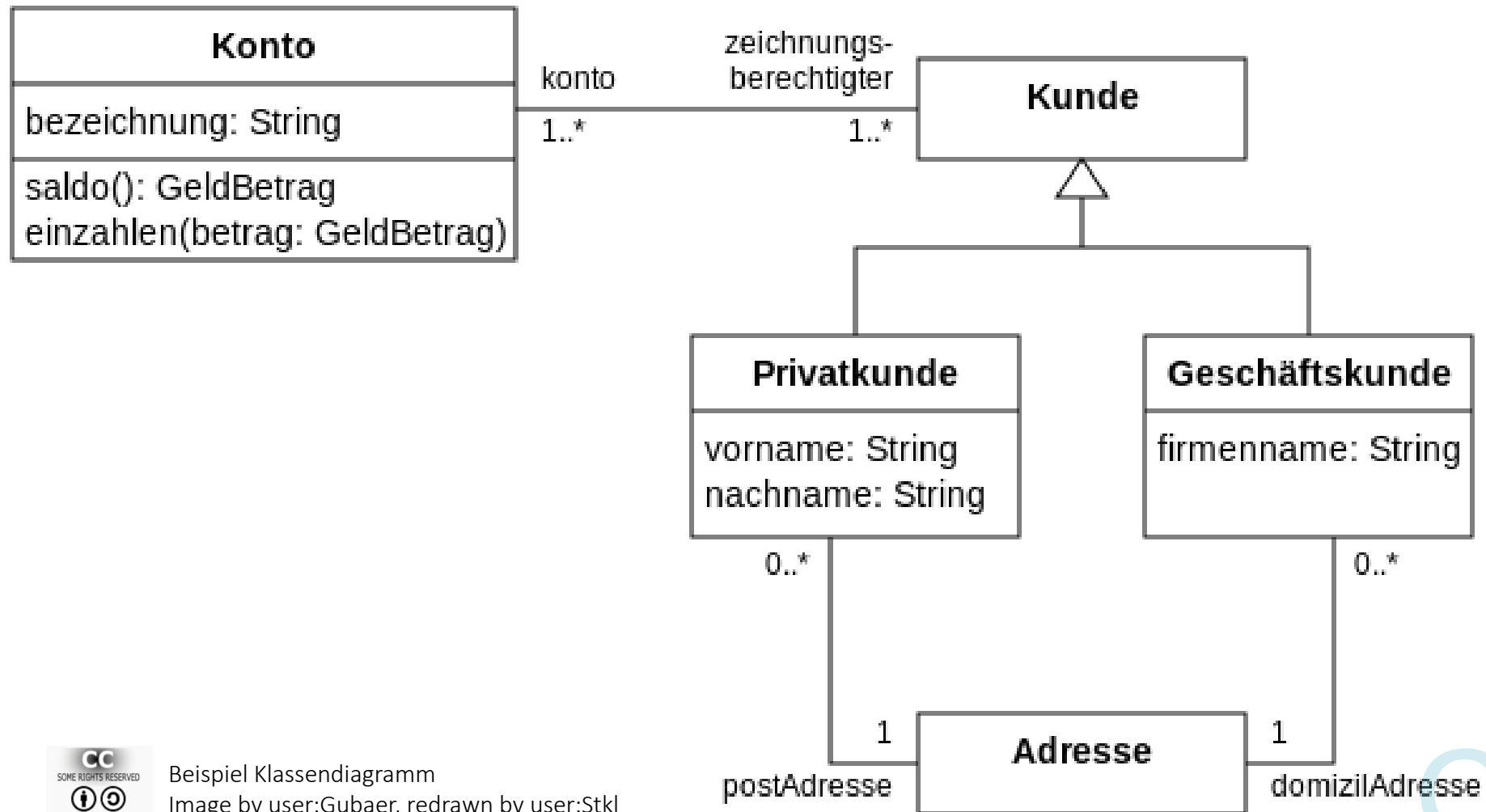


Which is more appropriate?

It depends on how the system should work



# Example



1. Classes and objects
2. Syntax and semantics
3. Advanced concepts

## Recap

Textbook chapters:

Systems Analysis and Design in a  
Changing World

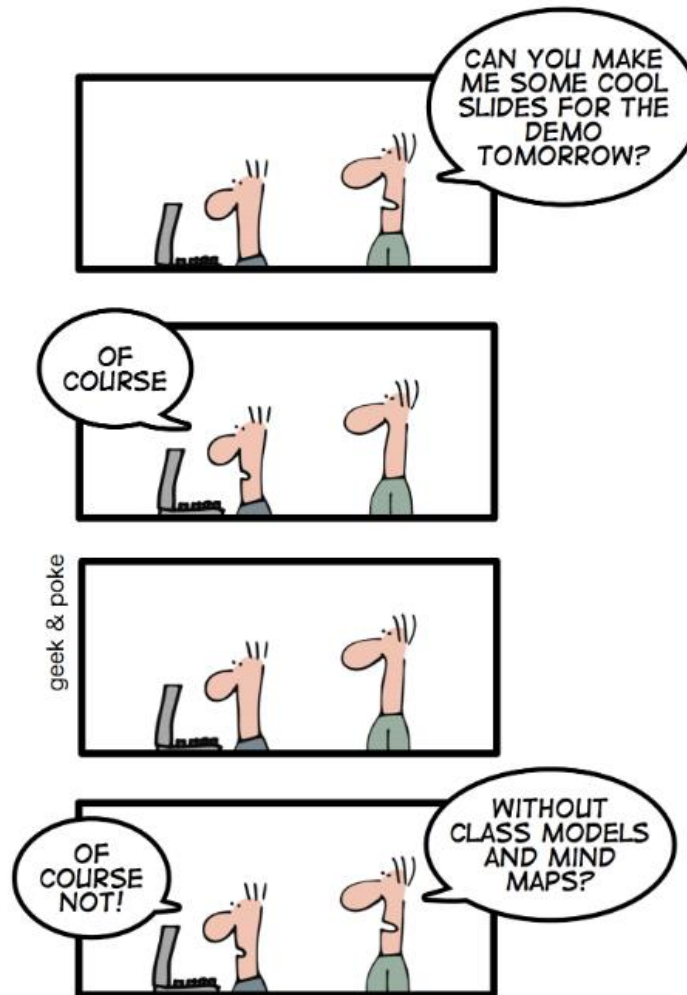
Chapter 4 Domain Modeling

Chapter 12 Object-Oriented  
Design: Fundamentals

UML@Classroom

Chapter 4 The Class Diagram

## CODERS AND SLIDES



Coders and slides

Image by Oliver Widder

<https://geek-and-poke.com/geekandpoke/2011/2/8/coders-and-slides.html>

## Class diagrams

11486 Systems Analysis and Modelling / 6677 Systems Analysis and Modelling G

Dr Luke Nguyen-Hoan